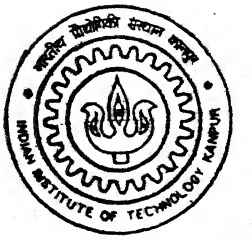
Domain Mapping with Minimal Distortion for Path Planning

By

S. PARTHASARATHY



DEPARTMENT OF MECHANICAL ENGINEERING

Indian Institute of Technology Kanpur

JULY, 2001

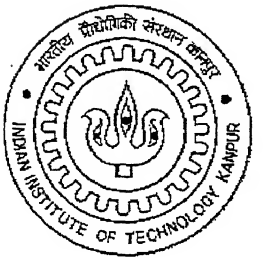
Domain Mapping with Minimal Distortion for Path Planning

A Thesis Submitted in Partial Fulfilment of the Requirements for the Degree of

Master of Technology

by

S. Parthasarathy



to the

DEPARTMENT OF MECHANICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR
INDIA

July, 2001

CERTIFICATE

This is to certify that the work under the thesis titled **Domain Mapping with Minimal Distortion for Path Planning** by **S. Parthasarathy (Roll No. 9910524)** has been carried out under my supervision and this work has not been submitted elsewhere for a degree.

Bhaskar Dasgupta

Dr. Bhaskar Dasgupta
Assistant Professor
Department of Mechanical Engineering
Indian Institute of Technology
Kanpur

Acknowledgement

I express my sincere thanks to my thesis supervisor Dr. Bhaskar Dasgupta for his excellent guidance, invaluable suggestions and constant encouragement during the entire period of this work. I am certainly fortunate to have him as my guide as he has been extremely patient providing me excellent guidance and support especially during the time when I was literally struggling with some of the problems in my thesis. He was always available at times of my need and has been quite understanding throughout.

I am highly thankful to Dr. Amitabha Mukerjee for enriching the fundamentals of CAD in me. I am grateful to Dr. N.N.Kishore for clearing my doubts in Finite Element Methods. I am also thankful to G.Saravanakumar, K.N.S.Sudheer and P.L.N.Prasad for helping me to develop some of the difficult algorithms in this thesis. I also thank Amit Gupta who helped me in using Genetic Algorithms.

I am thankful to all my batchmates for making my life at IIT-Kanpur memorable and enjoyable. I am particularly thankful to my labmates Tandon, Jaydeep, Pavan, Shamik, Koushik, Ankur and Bhutani for their constant support and encouragement. I once again thank K.N.S.Sudheer, Krishnamurthy, Neelakandan, Swaminathan and Sriram for providing me moral support during the course of my thesis.

And above all, I thank the Almighty for enabling me to achieve higher goals.

S. Parthasarathy

Dedicated to

My Beloved Parents

Contents

List of Figures

Abstract

There has been a renewed interest in path planning for robots in static workspaces as the environment is static in many applications. They are being extensively studied and solved since last two decades. The path required is a continuous sequence of points in the feasible region from the initial position to the goal position in a re-represented space called C-space, where the robot is represented as a point for each configuration. The optimality is more often judged on the length, safety from collision and geometric nature of the path.

In this thesis, attempt is made to plan a path which is close to optimum for a robot having 2 or 3 degrees of freedom. The workspace should be static and the C-space should be geometrically equivalent to a circle in 2-d or a sphere in 3-d. If the C-space is two dimensional, the initial region is mapped to a circle or if it is three dimensional, the initial region is mapped to a sphere and the obstacles in this C-space called C-obstacles (if any) are shrunk to points by minimally distorting the initial region in order to make the path close to optimum. This mapping is achieved in two steps. In the first step, the boundary of the initial region is mapped by minimizing its distortion. In the second step, the region inside is mapped by simultaneously shrinking all the C-obstacles to points inside this region again by minimally distorting the region inside. The initial and final positions between which the path has to be planned is connected by a straight line in this newly mapped region. This is remapped back to the original region to obtain the required path.

Chapter 1

Introduction

1.1 The task of path planning

The problem of path planning and static workspaces are being extensively studied and solved and there has been enormous developments starting from mid 1980's. Path planning is the way of making the robot decide automatically what motions to execute in order to achieve a task specified by initial and goal spatial arrangements of physical objects. In many applications the environment is static and the robot must perform a series of complicated maneuvers to achieve a sequence of goals. Recently there has been renewed interest in developing planner that can be applied to robots with many degrees of freedom (DOF's), say 6 or more.

The standard way to start solving the problem of path planning is to conveniently represent the robot, whose sequence of motions has to be decided, as a point. While this cannot be done in the actual workspace, the entire system is re-represented in another space where every configuration of the robot is a point and obviously the space will be of n dimensions, where n is the total degrees of freedom of the robot. This space is called the configuration space (C-space). The obstacles in workspace are correspondingly mapped to C-space and are termed as C-obstacles. The set of points in the C-space which are not contained by C-obstacles are feasible configurations of the robot in the actual workspace. The C-obstacles in C-space represent practically infeasible configurations which might be due to the presence of one or several of the following as shown in figure 1.1.

1. Work cell boundaries

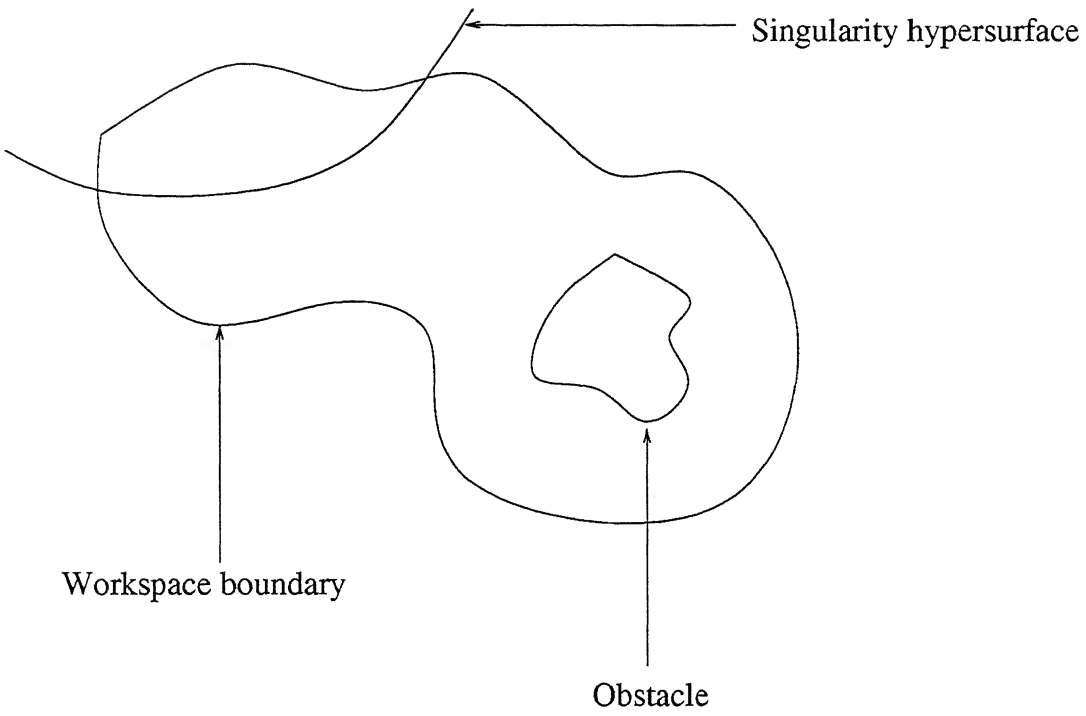
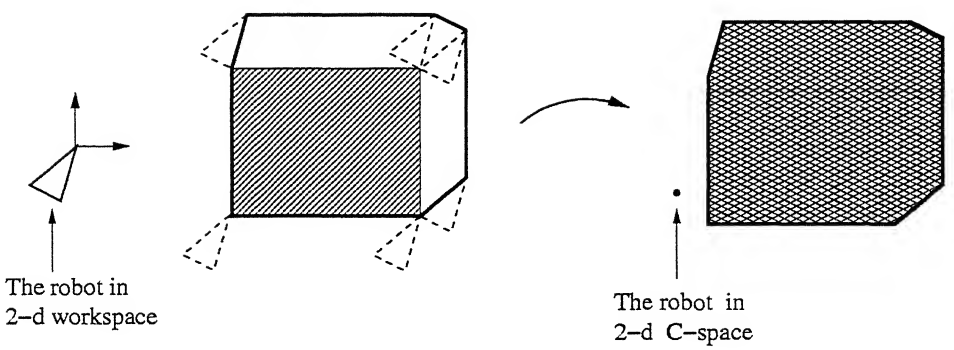


Figure 1.1: Robot's workspace

2. Singularity hypersurfaces

3. Obstacles

The underlying idea of configuration space is to represent the robot as a point in an appropriate space (the C-space) and to map the obstacles in this space. This mapping transforms the problem of planning the motion of a dimensioned object into the problem of planning the motion of a point [17]. A configuration of the robot is the specification of the position of every point in this robot relative to a fixed reference frame. A configuration can be described by a list of real parameters. Any configuration of the robot represented in the figure 1.2(a) as a triangle needs two real parameters to be described: the x and the y coordinate in its 2-d workspace. Figure 1.2(b) is the C-space where the robot is represented as a point with each point being different configuration of the robot. It is assumed that the robot represented in the figure 1.2(a) has translational degrees of freedom only.



The infeasible region (hatched) in 2–d workspace.

The infeasible region in 2–d C–space where the robot is a point.

Figure 1.2: The workspace and C-space in 2-d [17]

The task of path planning now has been reduced to finding a continuous sequence of points from the initial position to the goal position in such a way that this sequence should not intersect with any of the represented infeasible regions called as C-obstacles. This means that the path planned should be a continuous curve connecting two geometric points in this space, the initial and target positions. In summary, the problem is, to find a path for a point, in a non-convex region with a number of non-convex obstacles.

The next emphasis on this problem is to have an optimal path. The optimality is more often judged purely on lengths of the paths and safety from collision with obstacles. There are a number of other optimality criteria which are just as important if not more so, for certain applications. In particular determining paths of minimum turns is a very important problem in the area of robot motion planning. The complete enumeration of all the possible/feasible paths to establish an optimal one

is not feasible to implement as the search space increases exponentially (explodes combinatorially). This thesis aims at getting a path which is based on the length of the path and safety from collision against obstacles and boundary.

This thesis deals with the above problem of path planning in which configuration space is completely known. The initial non-convex region is transformed to a convex region without causing much distortion to the initial one. This procedure is called as "**Domain mapping with minimal distortion**". The domain mapping concept ensures the safety in terms of distance from obstacles and minimal distortion ensures to obtain a path of minimum turns. The entire C-space (the interior of C-space containing both free space and C-obstacles and the surface of the C-space) has to be transformed to a convex one with minimum distortion. This is achieved in two steps. The first step is to consider the external boundary of the C-space alone and to transform/map to another convex region of the same dimension with minimal distortion to it, so that a hollow convex region will be obtained and to establish correspondence between them. This step is called as "**boundary mapping**". The next step in domain mapping procedure is to map the inside points of the initial C-space region, corresponding to the new convex region obtained from the previous step. This step is called as "**inside mapping**". This inside mapping is also accomplished by minimally distorting the inside region. C-obstacles (if any) in the original region are shrunk to points inside the new convex region while mapping and the new position of these points will be obtained after minimizing the distortion. So the problem reduces to that of finding a path in a convex region where any two points can be connected by a straight line. Some simple strategy will be used to exclude the obstacle shrunk points on this line. If this case happens, the line is split in to two or more segments in such a way that none of the segments contain those points. Once a straight line or a series of straight lines is/are obtained, geometric shape functions are used to get a linear mapping to invert the entire path back to the original region by which the actual path will be obtained.

If a non-convex polyhedron in 3-d is transformed to a convex polyhedron with an isomorphic vertex neighbourhood graph, it is called as "**polyhedron realization**". This operation can be performed by setting a target convex regions as a base upon which our regions have to be mapped. As far as target regions are concerned, a circle in 2-d or a sphere in 3-d or a hyper sphere in n-d will serve the need. As the

distortion of the initial region has to be minimized while mapping, the distortion energy which is assumed to be zero in magnitude for the initial region, is taken as the degree of distortion underwent by the region after been mapped and the problem in both the steps is cast as an optimization problem minimizing this distortion energy.

Motion planning has enormous applications besides robotics. Some of them include computer graphics animation and molecular biology. Another very important application of motion planning is in medicine. Some of the newly developed surgical procedures extensively require motion planning.

1.2 Literature review

1.2.1 Optimal path planning problem

During the last three decades, motion planning has emerged as crucial and productive research area in robotics. Planning the path which humans do intuitively is not straightforward for planning for a robot. Starting from 1980's, conventional ways were considered to solve this problem. Although they gave considerably good results for lesser DOF's of robot, it became impractical to use those methods for higher DOF's. After the invention of high speed computational machines and with the emergence of evolutionary algorithms, path planning has entered a new era. Today planners efficiently deal with robots having many DOF's.

Some of the basic methods in path planning such as roadmap method, cell decomposition method, potential field method etc., are explained by Latombe [17].

Roadmap approach constructs a network of paths in free configuration space (C-space). This network is called the roadmap and represents standard paths. The problem reduces to that of joining initial and final position to these network. Roadmap methods include the "**Visibility graph (V-graph), Voronoi diagram and Silhouette methods**". All these three methods, compute in a single shot, a roadmap that completely represents the connectivity of free C-space. The V-graph and Voronoi diagram methods are limited to low dimensional C-spaces whereas the Silhouette methods applies to C-space of any dimension, but its complexity makes it less practical (Chung and Lee [3]). Improvements in these basic roadmap methods are found in literature. Approaches based on traversability vector, reduce the

data size and complexity of standard V-graph (Jason et al [10]). A "**Probabilistic roadmap planner (PRM)**" was explained by Kavraki et al [15]. PRM planner builds probabilistic roadmaps in the free C-space of the robot by generating and interconnecting a large number of random configurations of robot. These roadmaps are used for answering path planning queries: given an initial and final configuration of the robot, PRM planner connects them to the roadmap by simple paths and then searches the roadmap for a sequence of edges from one connecting node to the other. Theoretical basis for explaining its empirical success is given by Latombe et al [14]. The dependence of failure probability to connect an initial and final configuration on the length of a path, the distance function γ from the obstacles and the number of nodes N of the probabilistic roadmap constructed, is presented by Kavraki et al [13].

In potential field method, the robot, represented as a point in C-space, is moved under the influence of an artificial potential, produced by the goal configuration, workspace boundary and C-obstacles. The main limitation of this method is the presence of local minima in the potential fields. These minima may be difficult to escape. Local minima free potential functions (also called as navigation functions) have been defined by Rimon and Koditachek [21] but these functions are expensive to be computed in higher dimension C-spaces. Techniques for both computing potential functions and escaping local minima in higher dimensional C-space regions are given by Latombe et al [1]. The heat transfer analogy proposed by Wang et al [26] for path planning simulates steady state heat transfer with variable thermal conductivity. The path as planned in it is then the same as the heat flow with minimal thermal resistance.

Some planners combine the advantages of the above methods. The algorithm proposed by Warren and Charles [27] combines the desirable features of graph searching and potential field approaches. The algorithm proposed by Keerthi et al [16] introduces a heuristic roadmap method for path planning by extending the advantages from voronoi diagram concept. Here the roadmap is formed by connecting local maxima of a clearance function which is defined using distance functions. A developed version of PRM is used to plan the path by Bohlin et al [2] where the planner initially assumes that all nodes and edges in the roadmap are collision free, and searches the roadmap at hand for a shortest path between the initial and goal

node. The nodes and edges along the path are then checked for collision with obstacles. If a collision occurs, the corresponding nodes and edges are removed from the roadmap. Now this planner finds a new shortest path or first updates the roadmap with new nodes and edges, and then searches for a shortest path.

The algorithm proposed by Misashi et al [20] finds out one of the shortest paths using the analogy of wavefront of light, the theory put forward by Huygens in physics. The initial position is considered to be a light source which emanates light whose front will be spherical in nature at any instant. Once this collides with obstacles, they emanate secondary wavefront and some of them will reach the target position as it is assumed that the reflectivity is unity everywhere. A path is found so that the length from source to target is the least.

Motion planning with uncertainty for mobile robots is explained by Takeda et al [23]. Given a model of robot's environment, **"a sensory uncertainty field"** (SUF) is computed over the robot's C-space. At every configuration q, the SUF is an estimate of uncertainty (distribution of possible errors) in the "sensed configuration" that would be computed by matching the data given by the robot sensors against the environment model. A planner then makes use of the computed SUF to generate paths that minimize expected errors, so that their execution can be reliably monitored by the sensors.

New breakthroughs in the path planning have been achieved with the emergence of evolutionary algorithms and development in the field of soft computing. These advanced methods can be used to solve the problem of path planning effectively. Genetic Algorithms (GA's) are used extensively to plan the paths. The fitness function will be a function of several factors like goal factor, obstacle factor, smoothness factor, minimum path length factor, safety factor etc., of the path. The algorithm proposed by Dozier et al [5] combines fuzzy inference along with tournament selection to select candidate paths to be parents based on Euclidean distance from initial to goal position, sum of the changes in the slope of the path and the average change in the slope of the path. These GA's are quite good in planning paths for mobile robots. The planner proposed by Gerke and Michael [6] uses GA which calculates an optimal path according to a given optimization criterion. The path is planned by using GA whose fitness is a function of safety from obstructing dynamic objects and the distance to the goal position. The higher dimensional data is projected on

to lower dimensions to increase the speed of evaluation of the path as explained by Han et al [7]. Another new algorithm proposed by Vadekkeppat et al [24] combines artificial potential field method with GA's to derive artificial potential field functions. This is capable of navigating robot(s) situated among moving obstacles. GA is used as a search as well as optimization tool in the work explained by Juidette and Youlal [11]. Another algorithm proposed by Mearchou and Andreas [18], uses GA to determine near optimal path using a bit string encoding of selected graph vertices. The algorithm proposed by Sanderson et al [8] uses a multi-resolution path representation. If a successful path is found in the search hierarchy (at a low level of resolution), then further expansion of that portion of path search is not necessary. This advantage is mapped in to the encoded search space and adjusts the string length accordingly. The development of artificial potential field (APF) in potential field approach is computationally intensive. The realization that APFs can be developed by parallel distributed techniques has prompted interest in using neural networks for a least cost path planning. Vlassis and Esanakas [25] extends the idea of SUF explained above to plan paths in unknown and non stationary environments. A self organizing neural network model that is capable of building and maintaining estimation of SUF while the robot moves around its free space based on some dynamic localization information, is implemented.

1.2.2 Path planning and shape transformation

Metamorphosis or **morphing** is an interpolation technique used to create, from two objects, a series of intermediate objects that change continuously to make a smooth transition from the source to target configurations. This can be viewed as three distinct problems: correspondence, interpolation and feature specification. Correspondence refers to how points on one surface get mapped to points on the other. Interpolation is the actual transformation process. Feature specification refers to the capability of the user to control how features of the source object get mapped to features of destination object (Kent et al [9]). Yuan et al [28] also discusses new algorithms for geometric transformations.

Zhang et al [29] uses a method of mapping a 3-d non-convex region to a convex region and generating an artificial potential field in the mapped region.The stream-

line generated from initial position to target position in the artificial potential field gives the path. Suryawanshi et al [?] uses the idea of transforming a 2-d non-convex region, which is the given C-space, to some convex region like circle. The initial and target configurations in this new mapped space are connected by a straight line and the entire line is re-mapped back to the original C-space to get the required path. Here the mapped region will be close to circle. This method has been extended to 3-d C-spaces by Joshi and Dasgupta [19]. Here the initial non-convex region is mapped on to a sphere by initially discretizing the initial region by tetrahedrons. The features of the initial region are stored in terms of total layers of faces on this region and total faces in each such region. The destination region, namely the sphere, is made compatible with the features of the initial region mentioned above. Once the correspondence is established between the initial and target regions, the inside of the region is mapped by using finite element method. The mapped solid is a polyhedron whose topology approximately equals that of the sphere. To find a valid path between two points, they are joined by a straight line in the mapped region and inverted back to the original region.

This thesis attempts to tackle the problem of mapping non-convex 3-d region to a convex region by minimizing the distortion (which the region gets after it is mapped) by generalizing it for any non-convex region whose geometry can be approximated to a sphere. The final configuration of the mapped region in this thesis is the optimized solution obtained by minimizing the distortion energy. The positions of the mapped boundary nodes on the circle/sphere is not prescribed a priori as done in [22, 19], but is a solution for the optimization problem. To be more specific, the solutions obtained in [22] and [19] for 2-d and 3-d respectively are feasible solutions to the optimization problem, formulated in this thesis, while this thesis aims at obtaining an optimal solution to this problem.

1.3 Scope of the present work

- It is assumed that the C-space is well defined and the robot is represented as a point. The C-space should be 2-d or 3-d.
- For the sake of representing the C-space well, it is discretized by 4-noded

tetrahedrons which serve as the input.

- This thesis aims at mapping the non-convex (input) region to a sphere or a circle by minimally distorting it by formulating it as an optimization problem.
- The C-space can have any number of C-obstacles. Those obstacles will be shrunk to points inside the mapped region and their positions will be obtained after solving the formulated optimization problem to minimize the distortion energy with some important constraints.
- The geometry of the input C-space in 3-d should be equivalent to that of a sphere (i.e) it should be a zero-genus region.

1.4 Organization of this thesis

Chapter 2: Several algorithms to extract the desired data, which will serve as the input to the problem solver, from the supplied data are developed

Chapter 3: The procedure to map the boundary with minimal distortion is formulated.

Chapter 4: The procedure to map the inside of the original region to the mapped region, which is obtained from the previous chapter, is described.

Chapter 5: The procedure to plan the path is explained.

Chapter 6: Some of the results obtained by implementing all the algorithms described in the previous chapters, their inferences and related discussions are presented.

Chapter 7: Conclusions and future scope of this work are presented.

Appendix A: All the algorithms have been explained.

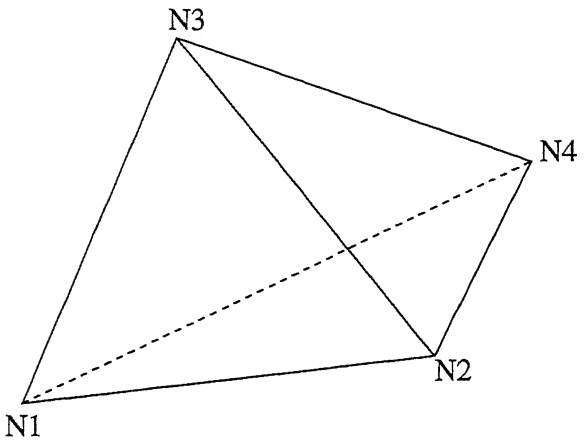
Chapter 2

The Preprocessor

The very first step in planning the path is, to represent the C-space in such a way, that the planner shall interpret this region quite comfortably to find the path from initial to final position inside it, entered by the user. In this thesis, the C-space is a 3-d region and is expected to be non-convex. This region is modelled in I-DEAS software. To get it better represented, this region is discretized to 4-noded tetrahedrons. The output file containing the data of the region modelled, cannot be used as it is, as the data is not explicit (the planner still needs more explicit data). This chapter explains various algorithms used to extract the data which the planner requires explicitly to plan the path. The responsibility of this preprocessing module does not end after extracting the data. This module is still needed to better define and re-process the extracted data. This is important because this data now becomes the input to the planner.

2.1 Requirements of the input to path planner

The output from I-DEAS, which serves as the input to path planner is called as universal (unv) file. This file contains the connectivities of all tetrahedrons after numbering every node. It also has the co-ordinates of each node. This data is quite raw, as it doesn't contain the required data for input to path planner explicitly. The way the universal file represents the region is shown in the figure 2.1. The input to path planner should explicitly contain the data required by the path planner which the unv file implicitly contains. The input to the path planner is available from the



4 noded tetrahedron.

N1, N2, N3, N4
. . .
. . .

N1
Xn1, Yn1, Zn1
.
.
.
N2
Xn2, Yn2, Zn2
.
.
.
N3
Xn3, Yn3, Zn3
.
.
.
N4
Xn4, Yn4, Zn4
.
.

Figure 2.1: The data in the unv file

preprocessor in the run time.

2.2 Conventions and declarations

- The connectivities of each tetrahedron is stored by initiating a linked list each node of which has a field of 4 integers to store 4 connectivities and from now on it may be called as type **tetrahedrons**.
- The connectivities of each triangle is stored by initiating a linked list each node of which has a field of 3 integers to store 3 connectivities and from now on it may be called as type **triangles**.
- The connectivities of each edge is stored by initiating a linked list each node

of which has a field of 2 integers to store 2 connectivities and from now on it may be called as type **edges**.

- The connectivities of each point is stored by initiating a linked list each node of which has a field of 1 integer to store the node number and from now on it may be called as type **points**.

- Every such types above, will also have flags, which will be set to one or zero as and when the necessity arises.

- In this report, the variable names will be represented in italics.

2.3 The algorithms

The algorithms presented in this thesis essentially use linked lists as they are very effective in handling the data and are quite fast in computation. These algorithms are implemented in preprocessor, which has a series of linked lists, which act as the data provider to path planner. The preprocessor obtains all the data regarding the tetrahedrons and stores them in the linked list called *alltetrahedrons*.

The first step is to extract all the triangles separately and store in a separate linked list. The algorithm to extract all the triangles has been presented in the appendix (algorithm-1).

The linked list obtained by implementing the previous algorithm contains all the triangles the region is composed of. The next step is to extract those triangles which lie on the boundary[1]. The basic idea to extract those triangles which bound the region is to find how many tetrahedrons a particular triangle share. If it turns out to be 1 then it is a bounding triangle. After extracting all the triangles, the bounding triangles are picked and are stored in a separate linked list by implementing the algorithm which has been presented in the appendix (algorithm-2).

The linked list obtained by implementing the algorithm-2 now contains all the triangles that lie on the boundary and the whole list will be meaningless if the

stitching, separates out the triangles belonging to the same boundary. This creates a total of *totalinternalboundaries* + 1 linked lists[2]. The algorithm to stitch has been presented in the appendix (algorithm-3).

All the triangles of the same boundary are picked up, stitched and stored in the array of linked list named *boundtriangles*, for all the boundaries. The next task is to identify the external boundary and store it in a separate linked list of type **triangles** called *extbound* and to delete it from the array of linked list *boundtriangles*. The algorithm to do this task is presented in the appendix (algorithm-4).

The external boundary and a node on the convex part of the region were identified. The path planner requires still more explicit data regarding the nodes on different boundaries, nodes which are inside the region, edges all over the region etc., precisely. The algorithms for this purpose which are implemented in preprocessor are explained in the appendix (algorithms 5, 6 and 7).

The total number of links in every linked list which are obtained up to now are calculated and each data is stored in separate variables or arrays as follows.

- Total number of nodes on the external boundary stored in *totalextnodes*.
- Total number of nodes on each internal boundary stored in array *totalintbdynodes* of size *totalinternalboundaries*.
- Total number of nodes that exist inside the region stored in *totalinnernodes*.
- Total number of triangles that exist on the external boundary stored in *totalexttriangles*.

2.4 The topology of the region

The next step in this preprocessing module is to re-define and re-process the extracted data. In addition to defining the **syntax** (the extracted data) of representations, we must also define their ***semantics*** (geometric meaning). The topology of the 3-d region (from now on, referred to as polyhedron in 3-d and polygon in 2-d), should be defined clearly by the preprocessor and will be transferred to the planner

[2]External boundary is also one among them.

along with the input, implicitly. This data will be used as the input by the path planner, along with the implicitly defined semantics, in the run time.

2.4.1 The boundary

So far, we have characterized a solid in its entirety; that is, as a 3-d region. We have only a description of the boundary. A proper connection between the topological structure of the entire region and the topological structure of its boundary should be made. This connection requires the concept of ***orientability***. The boundary must determine unambiguously what is "inside" and hence comprises the region under consideration. All the triangles on the boundary should be ***coherently oriented***.

What is this coherent orientation?

We orient a 2-simplex by cyclically ordering its vertices. These ordered connectivities are available in the linked lists. Figures 2.2(a) and 2.2(b) show the two possible orientations of the 2-simplex. Let S and S' be two adjacent 2-simplices in a complex C, and assume that they are adjacent in a 1-face S''. Then S and S'' are ***coherently oriented*** if the orientations of S'' induced by the orientations of S and S' are opposite. Figures 2.3(a) and 2.3(b) show two pairs of adjacent two simplices. The left pair is oriented coherently but the right pair is not.

What does this coherent orientation signify?

As already pointed out, the boundary should point inside and outside of the region unambiguously. As any plane in 3-d (triangle here) is referred to by its normal, we go for the direction of the normal. We will have to make the normal to each triangle, direct outward the region. Whenever the normal is calculated, we follow the conventions as shown in the figure 2.4. Every triangle has a direction of the normal associated with it for the given convention. If a node of the linked list *extbound* has the connectivities N1,N2,N3 then the normal is calculated as the crossproduct of the vectors N1N2 and N1N3. If all the normals of the triangles are on the external boundary, point outwards or inwards simultaneously, then all the triangles are oriented coherently.

The algorithm to orient all the triangles on the external boundary by making them coherent is explained in the appendix (algorithm-8).

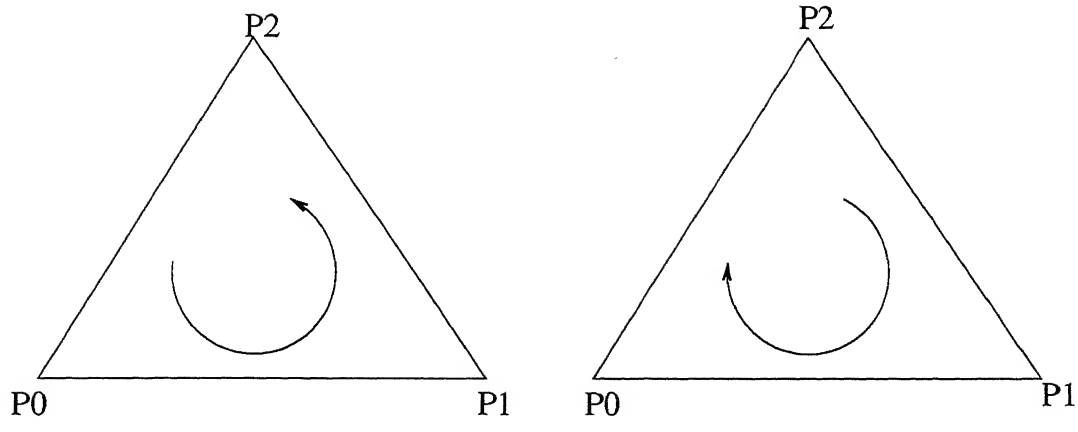


Figure 2.2: Orientation of a 2-simplex

2.4.2 The region inside the boundary

Once the external boundary is well defined, the region inside the boundary simply follows it, except the internal boundaries. Our aim is to shrink the internal boundaries to points and so the topology of internal boundaries is not brought to focus. The inside region takes the external boundary as its basis to get itself mapped. This will be discussed in chapter 4 in detail.

2.5 Two dimensional C-space

There is no major difference in using algorithms for 2-d C-space. The region which is defined as the C-space may henceforth be called a **polygon**. The given polygon will, in general be non-convex and may also have internal boundaries as well. If this C-space is also modelled in I-DEAS, the algorithms which were defined to extract the data needed by the path planner can very well be applied here also. The manifold of the highest dimension in case of 3-d was a triangle. Here it will be a line segment.

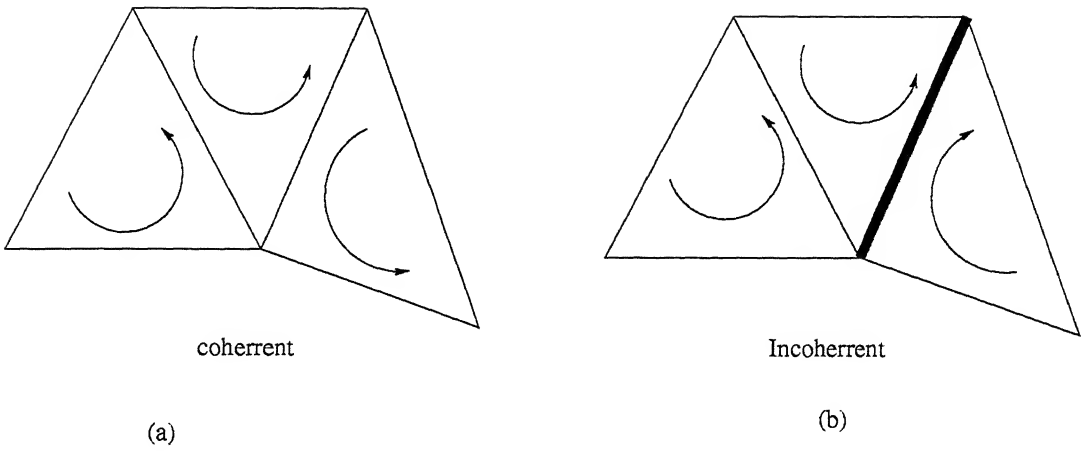


Figure 2.3: Coherent and incoherent orientations

The same linked list names will be defined while incorporating entities for 2-d C-space also.

2.5.1 The external boundary

While there is not much difference in defining the syntax, the semantics have to be re-defined. The semantics of the polygon is defined by the sense of its numbering on the external boundary on the basis of the concept of signed area. In right handed system, if a triangle is considered anticlockwise in numbering, then, the area of that triangle which is the half of the magnitude of the cross product of the two vectors considered as the side will yield another vector perpendicular to both the former vectors. If that normal vector is along the positive Z axis, then the area is positive or if that normal vector is along the negative Z-axis then the area is considered as negative.

Here in order to define the semantics, the polygon is numbered in anticlockwise direction as shown in the figure 2.5. That is, whatever connectivity is defined, when all put together, it should convey the sense that the polygon is numbered anticlockwise. A simple algorithm to change the connectivities in their order to make the polygon anticlockwise is explained below.

Starting from a node of any edge on the boundary of the polygon, one should be

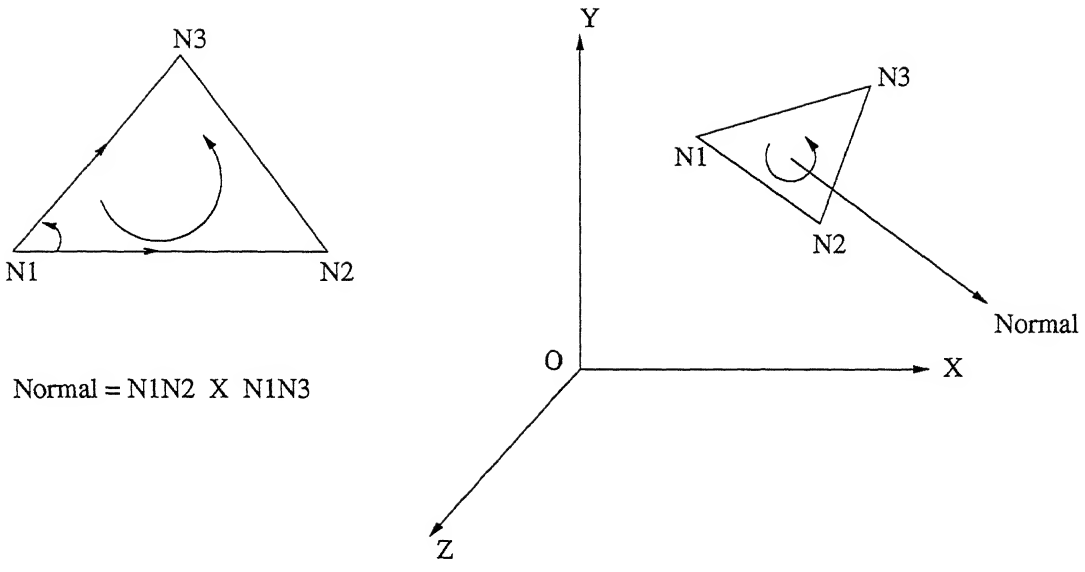


Figure 2.4: Significance of coherent orientation

able to traverse all the edges and come back to the starting node. The direction of the traversal is same as that of the sense in which the connectivities are stored. If it can be traversed, then all the edges are said to be coherently oriented. In figure 2.6(a), the edges are coherently oriented because if one starts at a node, keeping the sense same, one can come back to that node, but in figure 2.6(b) it is not so because if in a case one has started from node 1, then he has to stop at 3 which is a dead end according to the sense stored.

The algorithm to make the entire polygon anticlockwise has been explained in the appendix (algorithm 9).

2.5.2 The region inside the external boundary

The region inside the external boundary also uses the concept of signed area of triangles considered. Chapter 4 gives the algorithms in detail for mapping the interior region.

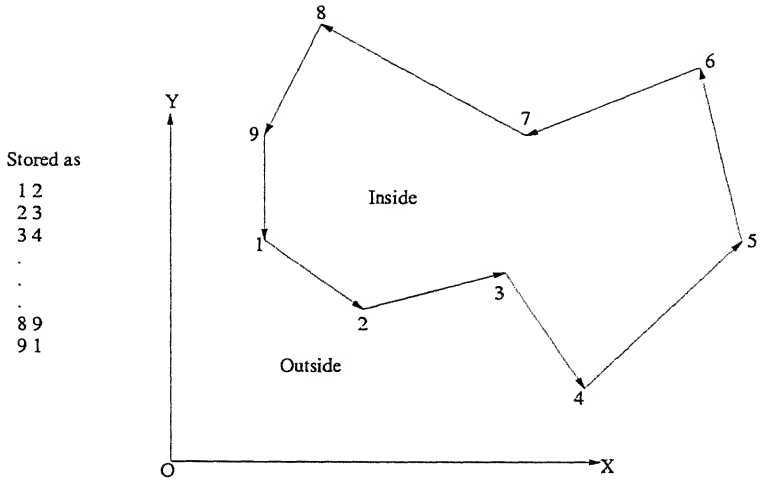


Figure 2.5: The anticlockwise polygon

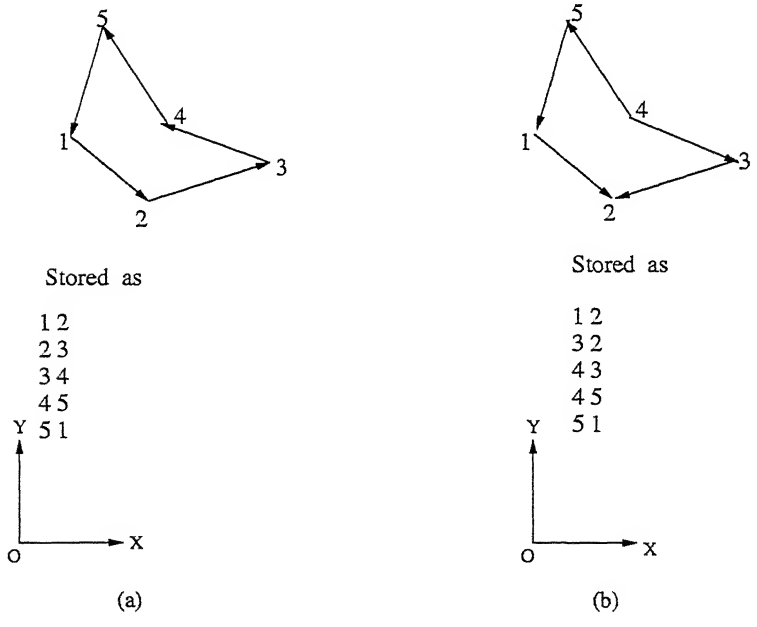


Figure 2.6: Coherent and incoherent orientation of the edges of a polygon

Chapter 3

Mapping the external boundary

The data, which is needed explicitly was extracted and the semantics of the initial region was also defined. This initial region has to be mapped to a sphere in case of 3-d or circle in case of 2-d by minimizing the distortion to the initial region. This implicitly assumes mapping both the entities, the boundary as well as the region inside this boundary, simultaneously. This will be obviously more difficult than solving two relatively simpler problems, which can be obtained by splitting the original problem. The first problem is to map only the external boundary without considering the region inside it. This chapter explains the procedure to map the external boundary alone without taking into consideration the region inside the boundary. The second problem, i.e. mapping the inside of the region, will be taken up in the next chapter.

3.1 The problem of boundary mapping

The problem now is to map the external boundary to a sphere (circle in case of 2-d) by minimizing the distortion of the boundary after mapping. The mapped polyhedron or polygon should be convex and minimally distorted with its nodes lying on the sphere or circle to which it is mapped.

The mapping of the boundary means, to get the final points of the nodes after they are mapped. If a single node is considered in space, total unknowns associated with it is 3 in 3-d or 2 in 2-d. It is better to represent all the nodes and hence all

the variables[1] in a single vector. That is, all the coordinates are to be assembled in a single vector. Presently, only the nodes on the external boundary has to be considered. If so, then those nodes has to be renamed from 1 to *totalextnodes*.

While the boundary is being mapped, the topology of the boundary is preserved by keeping the connectivities unaltered. That is, if a triangle A has neighbours B, C, D in the initial boundary, then that triangle will be bounded by the same neighbours after mapping. In the case of 2-d, an edge on the external boundary will have the same two neighbouring edges after mapping.

This problem is certainly difficult to solve. If for the better approximation of the initial region, the discretization level is increased, then the total tetrahedrons in 3-d or total triangles in 2-d and hence the total nodes all over the region are increased which increases the variables by 3 times in 3-d or 2 times in 2-d. The complexity of the problem, by increasing the total tetrahedrons or total triangles and hence total number of nodes, for better representation of the region, increases exponentially.

3.2 The optimization problem

The problem of this boundary mapping has to be cast mathematically to solve it. In order to cast it, certain assumptions are made which are listed below.

- Every triangle on the boundary is assumed to be linkages of springs of finite stiffnesses.
- Linear elasticity theory is assumed.
- Area of cross section of all the springs are assumed same

The objective function

The objective function is the total energy of all the springs, which exist on the external boundary, due to their deformation. It is given by

$$\mathcal{E} = \frac{1}{2}\sum_{i=1}^{n} k_i (\Delta l)_i^2 \tag{3.1}$$

where

[1]Total variables = 3 * totalnodes in 3-d or 2 * totalnodes in 2-d.

- n is the total number of edges on the external boundary.
- k_i is the stiffness of the i-th edge of the polyhedron or polygon.
- $(\Delta l)_i$ is the elongation of the i-th spring.

This objective function is quadratic in nature.

In addition to the constraints to make all the nodes of the mapped region lie on the sphere or circle, there should be some more constraints to ensure that the mapped region to be convex. If the latter constraints are not applied, a case of 2-d mapped region (a polygon) as shown in the figure 3.1 may be obtained because, the semantics defined for this polygon is not taken into consideration. In that case, the resultant polyhedron or polygon is not only non-convex but self-intersecting also, as shown in the figure 3.1. This is certainly an infeasible solution.

The constraints for ensuring the convexity of the mapped region should not merely sense whether a polygon or a polyhedron is convex or not, but also should sense the extent of convexity or concavity. That is, the constraints which are responsible for the desired output should be a continuous mathematical function rather a binary output function.

The polyhedron enclosed by a sphere

The intention of using the constraints is to get a convex polyhedron or polygon. As it was already noted, when the constraints were not applied, the resulting polyhedron was not only non-convex but also self-intersecting. This has no sense if it is used for planning the path. A circumscribed polyhedron has certain properties. Later in this chapter, these properties are modelled as the constraints. These properties are discussed below in this section.

The region is bounded by triangular patches. There are certain conditions which all the triangular patches should collectively satisfy. There are 2 such conditions. One of the condition uses the concept of area of a spherical triangle or the concept of solid angle.

The solid angle is the angle subtended in 3-d and is measured in sterdians. In order to subtend a solid angle, 3 vectors are necessary. They are subtended from a centre of the sphere. These three vectors create a spherical triangular patch at

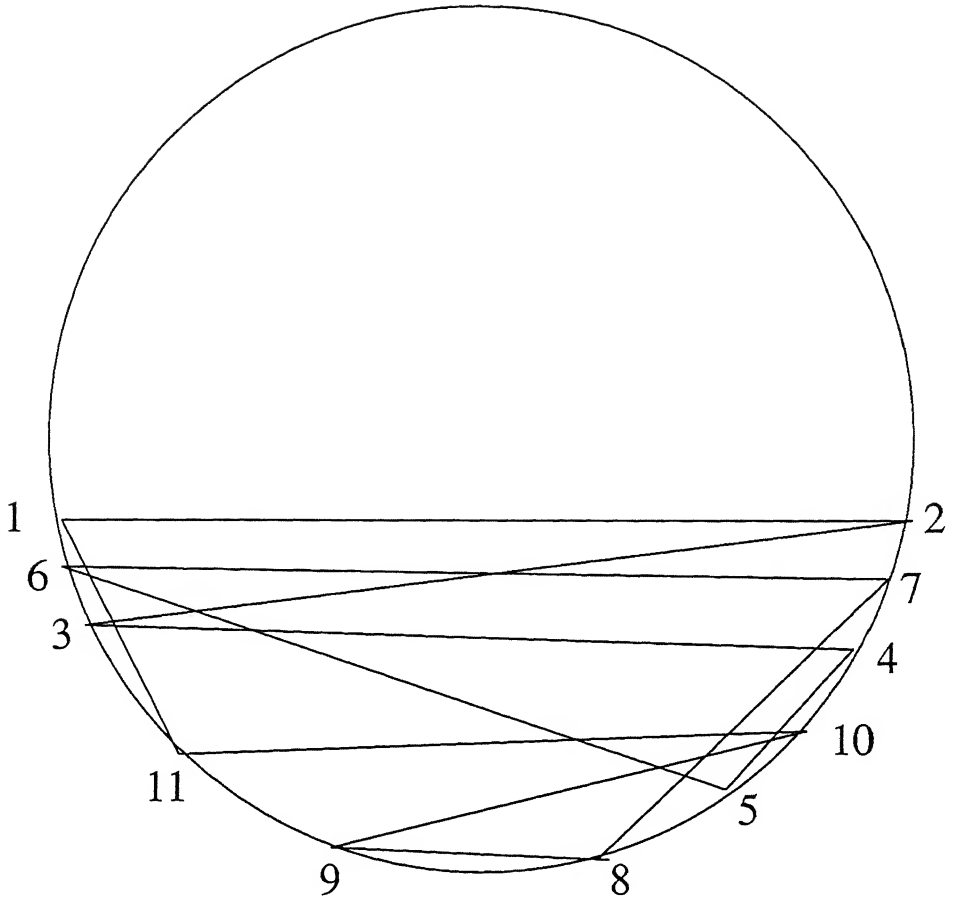


Figure 3.1: A polygon which is infeasible after mapping

the place where they intersect the sphere. As only the solid angle is considered, the radius is immaterial. Mathematically a solid angle is defined as follows.

$$Solidangle(E) = \frac{A}{(R)^2} \tag{3.2}$$

where

- A is the area of a spherical triangular patch which is calculated on the sphere of radius R

The idea of the solid angle and how it looks has been shown in the figure 3.2.

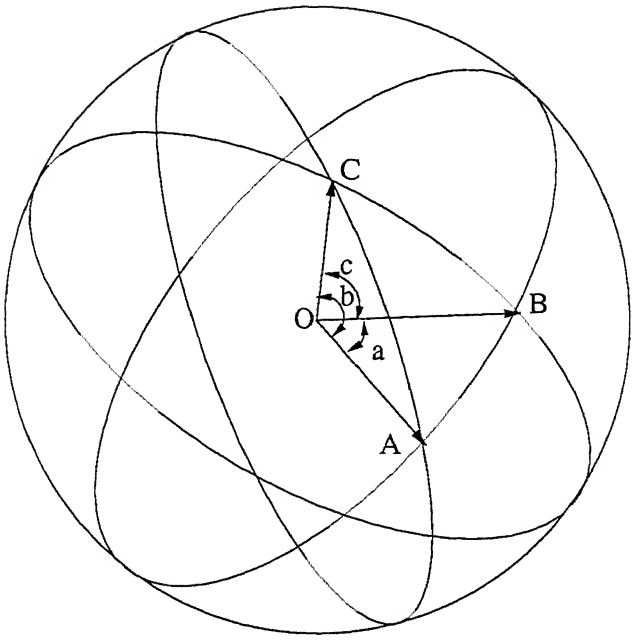


Figure 3.2: The concept of solid angle

The nodes of the triangle can lie anywhere on the vectors shown above, that is they can be outside or inside the sphere as shown in the figure 3.3. Mathematically the area of the spherical triangular patch and hence the solid angle is given below. Refer figure 3.2 for its geometric parameters.

For the case of simplicity, a sphere of unit radius is assumed.

$$E = 4 * \tan^{-1}\left[\sqrt{\tan\left(\frac{s}{2}\right)\tan\left(\frac{s-a}{2}\right)\tan\left(\frac{s-b}{2}\right)\tan\left(\frac{s-c}{2}\right)}\right] \tag{3.3}$$

where

- a,b,c are the angles between each of the three vectors mutually.
- $s = \frac{a+b+c}{2}$

The calculation of solid angle has certain special cases. They are discussed below.

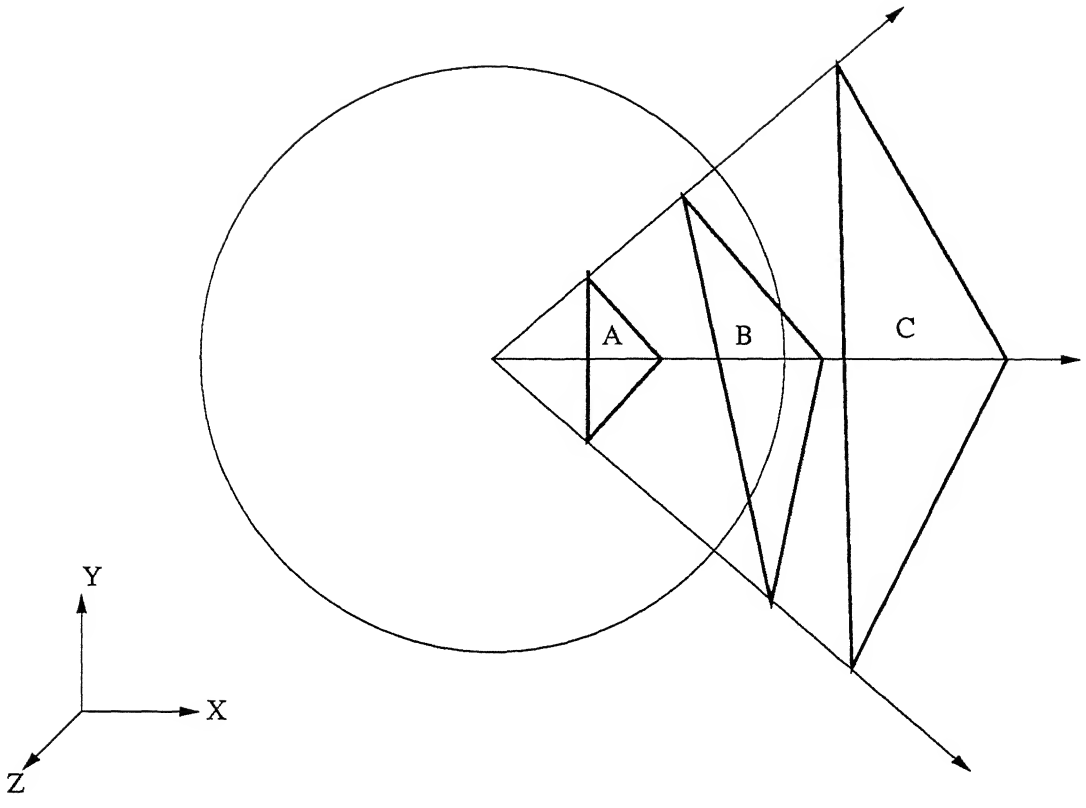


Figure 3.3: Same solid angle for different planar triangles

- If the plane containing the triangle whose solid angle has to be measured contains the centre of the sphere and if that centre is well bounded by three edges of the triangle as shown in the figure 3.4(a), then mathematically the solid angle is not well defined. But in our case it is assumed as 2π sterdians.

- The other special case is one where the centre of the sphere may almost coincide with any one of the nodes of the planar triangle in consideration or it may so happen that the centre may well lie outside the triangle but lying on the plane defined by the triangle as shown in the figure 3.4(b). In both the cases, solid angle is not well defined and they are assumed as 0 sterdians in this thesis.

The total solid angle of any sphere is 4π sterdians from the definition of the solid angle. This is the first property. Here in this problem, solid angles are calculated

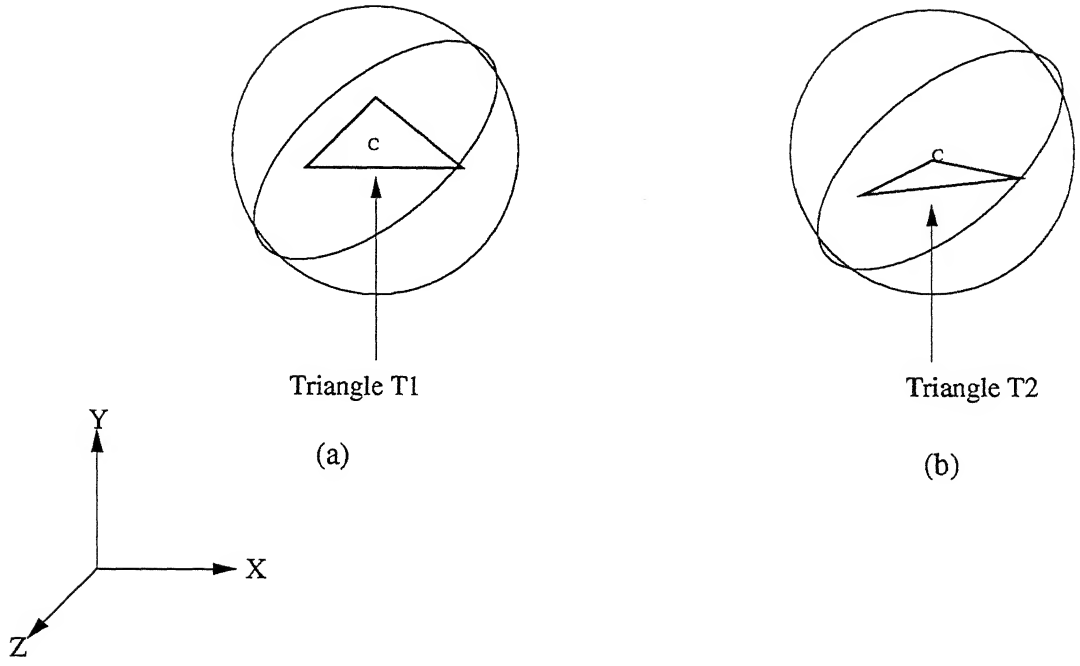


Figure 3.4: Special cases in solid angle calculation

for each triangle and finally summed up. The sum has to be 4π sterdians. While finding the solid angle of each triangle, a new designation is given to each of them i.e. whether they are major or minor. This is found by their sense[2] or orientation. The major triangles will have solid angles greater than or equal to 2π and minor triangles will have solid angle less than 2π sterdians.

If once the semantics are well defined for the polyhedron, then, for the polyhedron to be convex, the angles between each triangle with its neighbours, subtended about the sharing edge, inside the polyhedron as shown in the figure 3.5, should be less than or equal to π radians.

These two properties will not only give the result whether the polyhedron is convex or infeasible, but also the degree of its convexity or infeasibility. These two properties go hand in hand and so they have to be applied simultaneously.

[2]This sense is the geometric meaning held by a particular triangle while meaning the whole polyhedron. This is elaborated in the sub section 3.2.1.

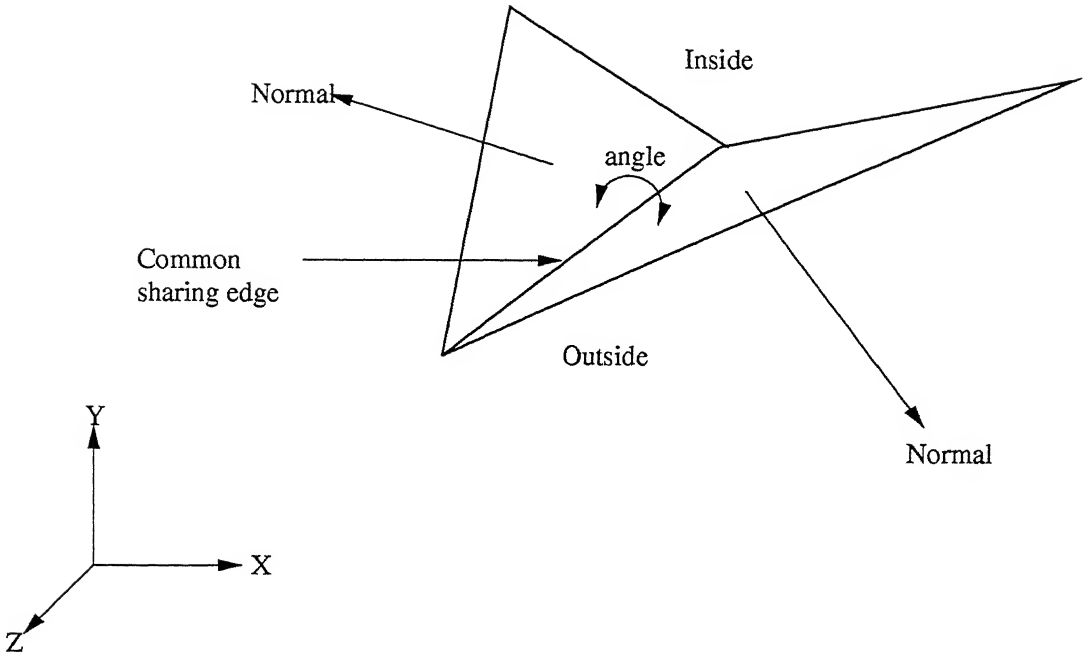


Figure 3.5: The inside angle of two triangles

The circumscribed polygon

A circumscribed polygon also has two properties which will be applied as constraints. Those properties give out the degree of satisfaction or violation with the nature of the function of these constraints being continuous.

The semantics of the polygon is well defined by naming the polygon anticlockwise. The total angle of any circle is 2π radians. For a circumscribed polygon, each edge has an angle associated with it. This angle will be negative if that edge is a major edge and positive if it is minor. The concept of minor and major edges are explained in constraints subsection in detail. All angles are then converted to positive quantities and their sum should be 2π radians. This is the first property.

The second property is that the triangles formed by neighbouring edges of the external boundary should have positive signed area i.e. the triangle formed should be an anticlockwise in sense. The nodes of the edges have to be taken in the same way it were stored. This is shown in the figure 3.6.

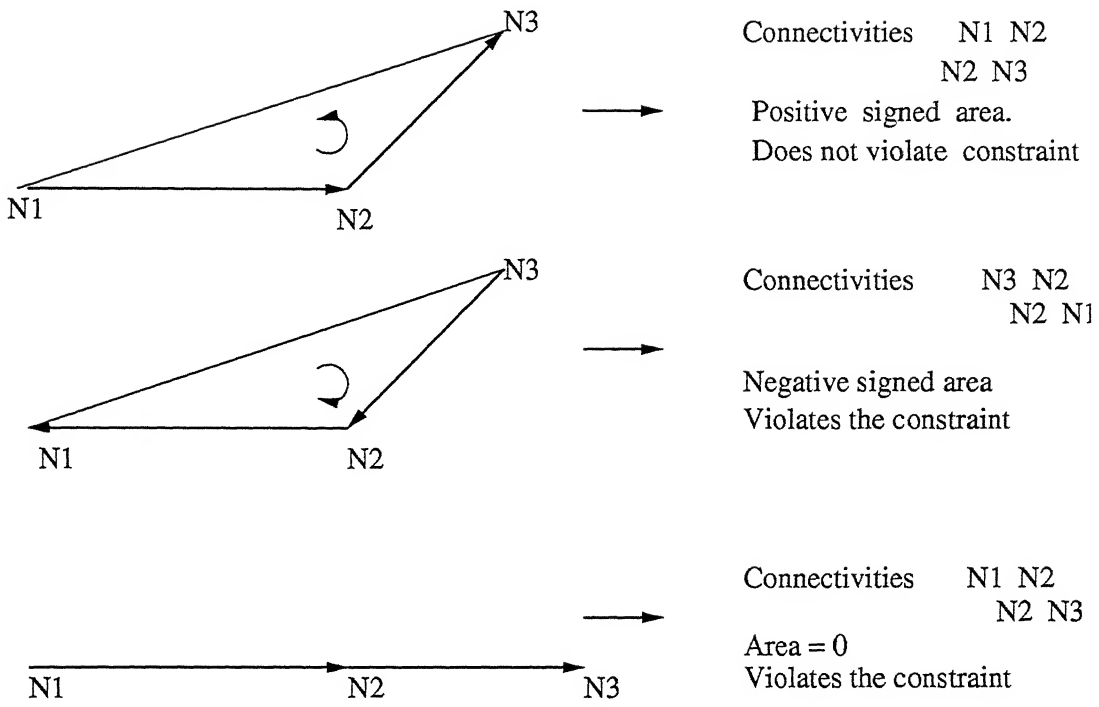


Figure 3.6: Signed area of the neighbouring edges

3.2.1 Equality constraints in boundary mapping

1. **Constraints to ensure that all the nodes on the external boundary should fall on the sphere after mapping.**

   The centre and radius of the sphere are calculated as follows.

   The centre of gravity of the initial region is calculated as

$$Xc = \frac{\sum_{i=1}^{N} V_i X g_i}{\sum_{i=1}^{N} V_i} \tag{3.4}$$

$$Yc = \frac{\sum_{i=1}^{N} V_i Y g_i}{\sum_{i=1}^{N} V_i} \tag{3.5}$$

$$Zc = \frac{\sum_{i=1}^{N} V_i Z g_i}{\sum_{i=1}^{N} V_i} \tag{3.6}$$

   where

- Xg,Yg,Zg with suffix i represents coordinates of centroid of each tetrahedron.
- V_i represents volume of i-th tetrahedron.
- N represents total number of tetrahedrons all over the region.

The centre of gravity calculated above becomes the centre of the sphere.

In the case of 2-d , the centre of gravity of the initial region is calculated as follows.

$$Xc = \frac{\sum_{i=1}^{N} A_i Xg_i}{\sum_{i=1}^{N} A_i} \tag{3.7}$$

$$Yc = \frac{\sum_{i=1}^{N} A_i Yg_i}{\sum_{i=1}^{N} A_i} \tag{3.8}$$

where

- Xg,Yg with suffix i represents coordinates of centroid of each triangle.
- A_i represents area of i-th triangle.
- N represents total number of triangles all over the region.

The centre of gravity now calculated becomes the centre of the circle.

The initial region has to be mapped to a sphere with minimal distortion. If a radius is chosen arbitrarily for that sphere, the initial region will acquire an energy which is responsible for increase or decrease in size, called as scaling. Distortion energy means, the energy for altering the shape only. Hence the radius can not be chosen arbitrarily, instead, the radius is calculated as the radius of that sphere whose volume is equal to the volume of entire region before mapping. In the case of 2-d, the radius is that radius of a circle whose area is equal to the area of the entire polygon before mapping.

The following models the constraints which ensure that every node on the external boundary should lie on the sphere or circle.

For i = 1 to N

$$h_i = (x_i - Xc)^2 + (y_i - Yc)^2 + (z_i - Zc)^2 - Rad^2 = 0 \tag{3.9}$$

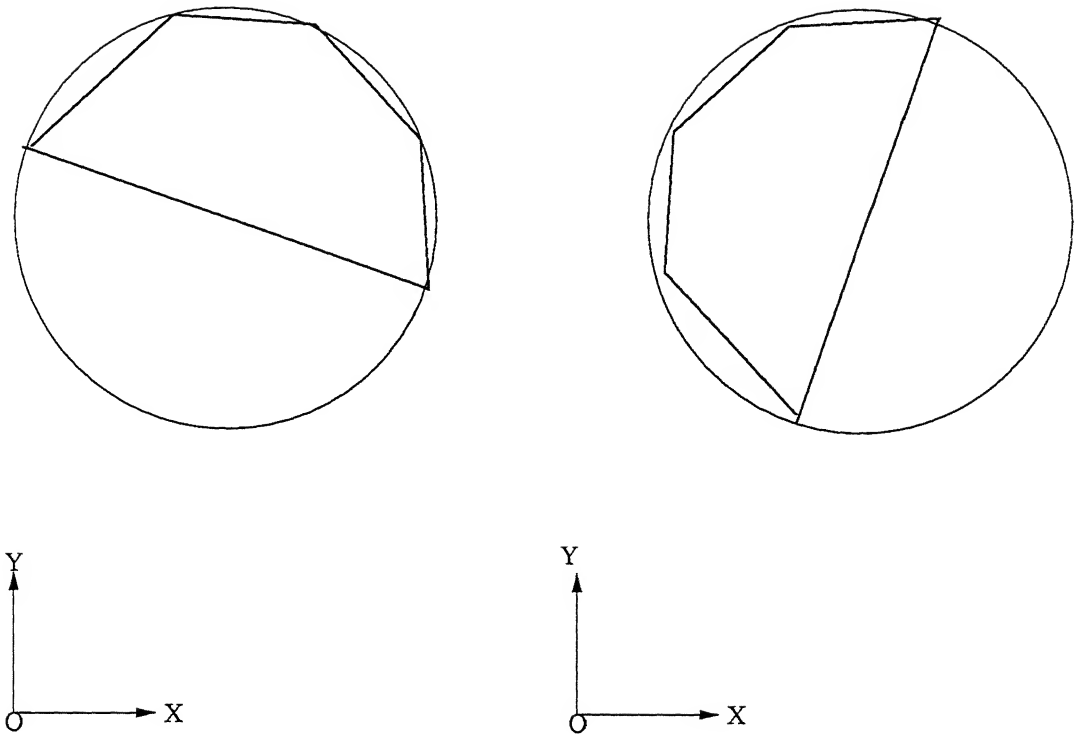


Figure 3.7: The mapped polygon sliding inside the circle

where

- N is the total number of nodes on the external boundary
- x_i, y_i, z_i are the coordinates of i-th node in space.
- Rad is the radius of of the sphere to which the polyhedron is mapped.

In case of 2-d, the term involving Z, (z_i-Zc) vanishes in the above constraint and Rad is the radius of the circle to which the polygon is mapped.

2. **Constraints to break the symmetry of the resultant polyhedron.**

   The resultant polyhedron can slide along the inner surface of the sphere. This case is schematically represented in the figure 3.7 for a 2-d case. Every such orientation is solution, i.e. if this constraint to restrain sliding along is not applied, the problem has infinite solution. Even though we are not bothered

about where the polyhedron is mapped inside the sphere, so for as optimization problem solver is concerned, this constraint has to be applied to make the problem have a definite solution. The following steps are involved in applying this constraint.

- We already found the *outmostnode*. Another node which is neighbouring to it is found.
- The plane containing the nodes above and the centre of the sphere in the initial region is defined by the plane's normal. This normal is found by certain type of orientation of the nodes.
- The normal of the plane defined by the same nodes and by the same orientation followed, is calculated for the mapped polyhedron.
- The direction cosines (dc's) are calculated for both the cases (before and after mapping) and they are equated to get the equality constraint. If the normal in the mapped polyhedron is not defined[3] then the direction cosines can well be assumed to be 0.

Mathematically the constraints are modelled follows.

$$h_i = \cos\alpha_1 - \cos\alpha_2 = 0 \tag{3.10}$$

$$h_{i+1} = \cos\beta_1 - \cos\beta_2 = 0 \tag{3.11}$$

$$h_{i+2} = \cos\gamma_1 - \cos\gamma_2 = 0 \tag{3.12}$$

where

- Suffix 1 denotes dc's of the plane before mapping
- Suffix 2 denotes dc's of the plane after mapping.

In the case of 2-d, the constraint is applied as follows,

- Find out a vector from the centre of the circle to the outmost node on the external boundary with the initial coordinates. Let this be $\vec{A}$.

[3]In this case, there will be no plane defined by the nodes in consideration.

- Find out another vector from the centre of the circle to the same outmost node but with the mapped coordinates. Let this be $\vec{B}$.
- Find out the unit vector of $\vec{A}$. Let this be written in the form $l_1\vec{i} + m_1\vec{j}$
- Find out the unit vector of $\vec{B}$. Let this be written in the form $l_2\vec{i} + m_2\vec{j}$
- If the unit vector of $\vec{B}$ is not defined, then l_2 and m_2 can be assumed as zeros.
- The constraints are modelled mathematically as follows.

$$h_i = l_1 - l_2 = 0 \tag{3.13}$$

$$h_{i+1} = m_1 - m_2 = 0 \tag{3.14}$$

where

  - Suffix 1 denotes dc's of the plane before mapping
  - Suffix 2 denotes dc's of the plane after mapping.

3. **Constraint to ensure convexity of the resultant polyhedron.**

Major and minor triangles

Every triangle in the polyhedron is designated by a keyword major or minor. A total of 4π sterdians should be obtained if all the solid angles are summed up. This summing up of solid angles is algebraic. The solid angle is supposed to be negative if the dot product of the normal of the triangle and the vector from the centre of the sphere to any one of the nodes on the triangle is negative. For our simplicity, the triangle is considered to be major even if the dot product is considered 0. This means that the centre of the sphere lies on the plane defined by the triangle. The negative solid angle is converted to positive one by adding it with a value 4π. So for all the major triangles, the solid angle will be greater than or equal to 2π sterdians. The concept of minor and major triangle is explained in the figure 3.8.

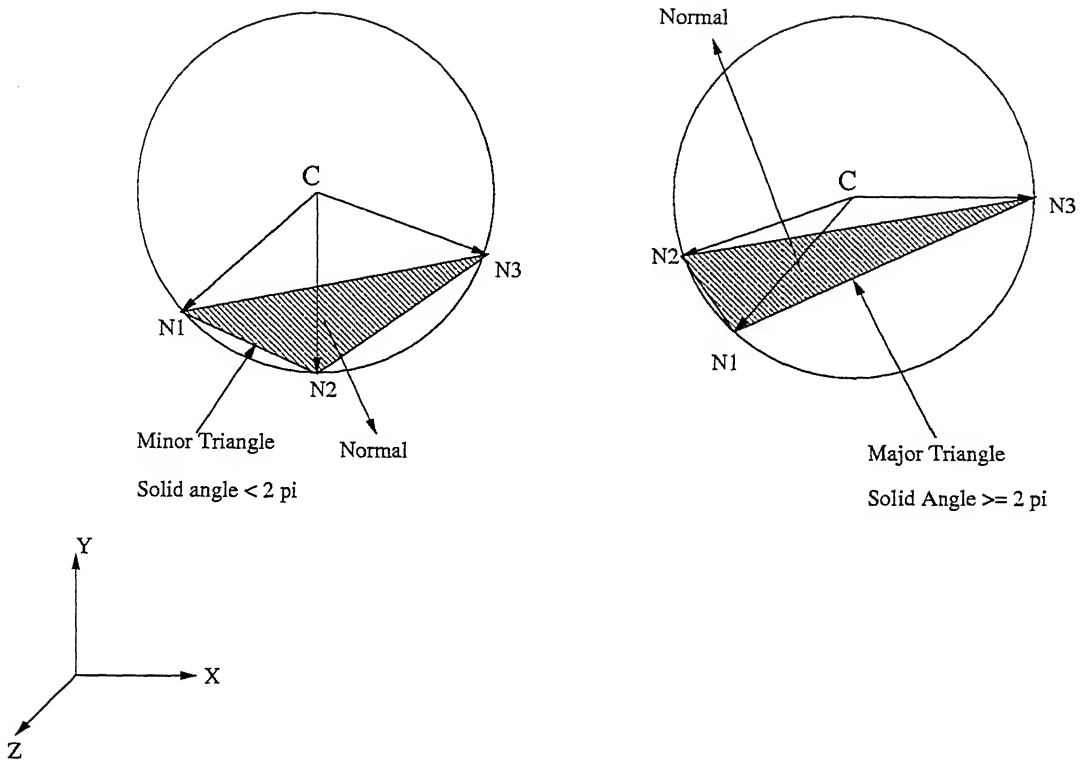


Figure 3.8: Minor and major triangles

Major and minor edges

Every edge in the polygon is also designated by the keyword major or minor. A total of 2π radians should be obtained, if all the angles of each edge are summed up. This sum is an algebraic sum. For major edges, the angle what they subtend will be greater than or equal to π radians. The figure 3.9 shows how they subtend the angle. If for the same convention used for a particualr edge, if the signed area of the triangle defined by the centre of the circle and the two nodes of the edge is negative, then the angle subtended by this edge will be the reflex angle between the lines joining the centre of the circle and the other nodes of the edge of the polygon. Else, the angle of the edge is the smaller angle between the same lines. A sum of all such angles for all edges should be 2π radians.

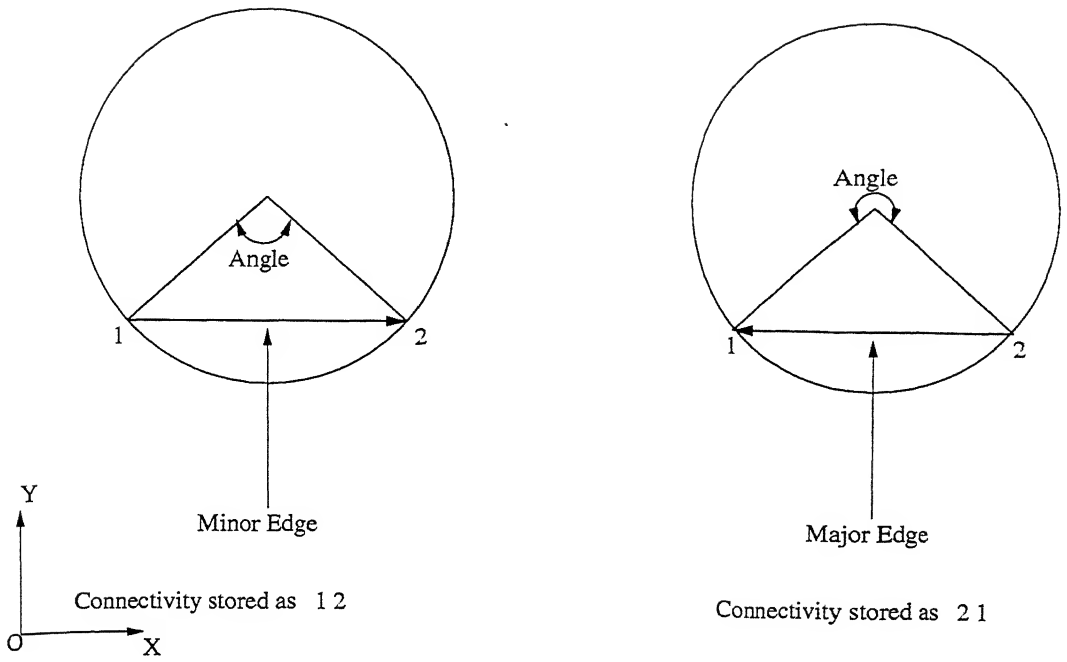


Figure 3.9: Minor and major edges

The algorithm to calculate solid angles for each triangle on the external boundary in the case of 3-d is detailed in the appendix (algorithm 10).

The constraint is now coded mathematically as follows.

$$h_i = \sum_{i=1}^{N} solidangle_i - 4\pi = 0 \tag{3.15}$$

where

- N is total number of triangles on the external boundary.
- $solidangle_i$ is the solid angle of i-th triangle on the boundary of the polyhedron

The algorithm to calculate the angle of each edge of the polygon in case of 2-d is elaborated in the the appendix (algorithm 11).

The constraint is coded mathematically as follows

$$h_i = \sum_{i=1}^{N} edgeangle_i - 2\pi = 0 \tag{3.16}$$

where

- N is total number of edges on the external boundary of the polygon.
- $edgeangle_i$ is the edge angle of i-th edge on the external boundary of the polygon.

3.2.2 Inequality constraints in boundary mapping

All the constraints modelled in this section are of the type greater than or equal to zero.

1. **Constraints to ensure the convexity of the resultant polyhedron after mapping.**
   These are the constraints which have to be applied along with the previous set of equality constraints to ensure the convexity of the resulting polyhedron.

   In order to apply these constraints, we have to have all the combinations of 2 triangles on the boundary which share their edge. The following algorithm extracts out all the triangles which share an edge on the boundary.

   The algorithm to extract all the triangles which share an edge on the boundary is detailed in the appendix (algorithm 12).

   Care is taken while storing the connectivities of the triangles. The order of storage is not disturbed. This always makes the normal to point outside the region as discussed previously.

   The idea of the previous algorithm is again used to extract the neighbouring edges in case of 2-d. They are stored with the name *combinations* which is capable of storing two edges. The connectivity or the sense here too will be maintained.

   The following steps are followed to apply this constraint in 3-d.

   - Start traversing the linked list *combinations*

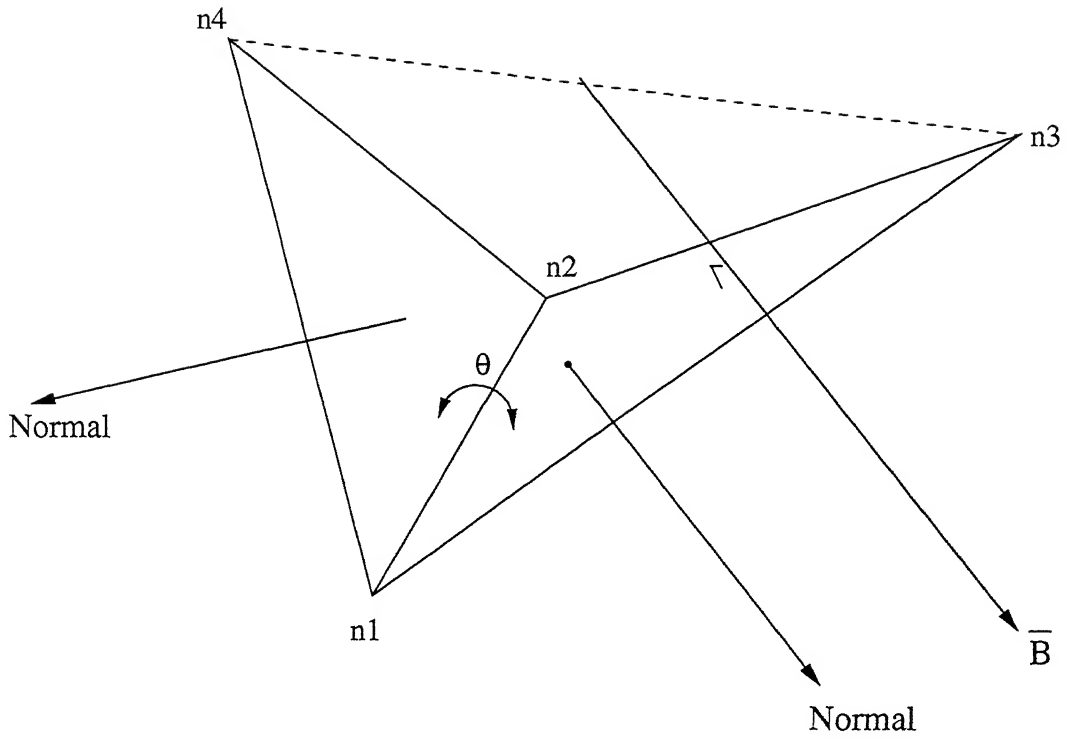


Figure 3.10: Calculation of the included angle

- For each node of this linked list, there will be two triangles sharing one common edge. Let the nodes corresponding to the edge shared by two triangles be *n1* and *n2*. The other two nodes shall be *n3* and *n4*.
- Let the normal of the triangles as per convention be $\vec{N}1$ and $\vec{N}2$ for the triangles *triangle1* and *triangle2* respectively.
- Find out another vector from the midpoint of the line segment joining nodes *n3* and *n4* to the point where this vector cuts any one of the triangle (say *triangle1*) perpendicularly as shown in the figure 3.10. Let this vector be $\vec{B}$.
- Find the dot product of the vectors $\vec{B}$ and $\vec{N}1$.
- Find out the angle between the triangles *triangle1* and *triangle2* which they subtend at the sharing edge.

- If this angle between the triangles is not equal to or not almost equal to π radians, then
  - If the dot product is negative, then the angle calculated is negative. It can be made positive by subtracting it from 2π radians.
- The condition of dot product having 0 value shall be made returning an infeasible value.
- This angle be θ and be stored in each node of the linked list *combinations* which contains a couple of triangles.

The angle θ calculated in the previous step for each node in the linked list *combinations* is the angle subtended by two triangles in combination inside the polyhedron. Hence for the polyhedron to be convex, this angle should not exceed π radians. This is the basic idea behind this constraint.

This constraint is mathematically modelled as follows.

$$g_i = \pi - \theta_i \geq 0 \tag{3.17}$$

where

- θ_i is the angle subtended by the two triangles calculated according to the convention which they subtend at the sharing edge for i-th node of the linked list *combinations*.

The total number of such constraints will be the total nodes in the linked list *combinations*.

The following steps are followed to apply this constraint in 2-d

- Start traversing the linked list *combinations* of type *edges* for 2-d.
- For each node of this linked list, there will be two edges sharing one common point. Let the node shared by both of them be *N2*. The other nodes shall be *N1* and *N3* provided the sense by calling the nodes *N1*, *N2*, *N3* should comply with the way they are stored.

- Find out the signed area of the triangle with the order *N1,N2,N3*. Let the area be A.
- Now the constraint can be modelled mathematically as

$$g_i = A - \epsilon \geq 0 \tag{3.18}$$

where

- ϵ is a lower limit usually around 10^{-6}, on the value of area.

2. **Constraints to ensure that none of the triangles after mapping converge to a point, in 3-d or none of the edges on the external boundary should fully get compressed in 2-d.**

There should be some more constraints which should restrain each spherical triangle from getting zero area after mapping. This may very well happen to some polyhedron geometries if the constrained minima would be obtained for full compression of any one of the edges on the external boundary. The constraint is applied by forcing each spherical triangular area to be not less than certain minimum value which shall be computer defined[4] zero.

These constraints are essentially of the type greater than or equal to zero. They are modelled as follows.

$$g_i = A_i - \epsilon \geq 0 \tag{3.19}$$

where

- A_i is the spherical triangular area of the i-th triangle on the external boundary.
- ϵ can be any small positive floating point number.

In the case of 2-d, a lower minimum limit on length can be fixed for each edge by applying the following constraint.

[4]This case is popularly known as the epsilon case.Here the zero for the computer is defined as some small floating point number such as 10^{-6} or so.

$$g_i = L_i - \epsilon \geq 0 \tag{3.20}$$

where

- L_i is the length of the i-th edge on the external boundary.
- ϵ can be any small positive floating point number.

3.3 Solving the optimization problem

In order to solve this constrained minimization problem, some numerical optimization procedure has to be used. The objective function is quadratic whereas some of the constraints are inverse trigonometric functions. The main difficulty which arises while solving this problem is the difficulty to trace the search space. The difficulty increases exponentially with increasing number of nodes. This is very difficult to solve (impossible in certain cases) by conventional optimization procedures because they need very good guess.

This necessitates the use of evolutionary optimization procedure called Genetic Algorithms (GA) to solve this problem. If the operators in GA are problem specific then GA will yield a solution close to the exact. Mostly GA will give most probable solution, whose degree of feasibility will be high or in other words, that solution will be a good guess. In this thesis, the solution given by GA will be taken as the initial guess and later conventional optimization procedures will applied to get the accurate results. Real coded GA using simulated binary crossover (SBX) operator is used in this thesis. Much of theory on GA is available in Goldberg [4] and theory regarding GA with SBX crossover operator is available in Deb and Beyer [12].

In this thesis, the problem is to minimize the distortion energy satisfying all the constraints. A fitness function is defined in GA. This is higher for certain member in a population if it satisfies more number of constraints and also keeps the distortion energy lower than other member.

Mathematically, in order to calculate the fitness, the constrained optimization problem is converted into unconstrained optimization problem by adding the violations of the constraint, each magnified by a constant penalty parameter to the objective function value.

If the problem is

$$E = f(x_1, x_2, ..., x_n) \tag{3.21}$$

such that

$$h(x_i) = 0 \tag{3.22}$$

$$g(x_j) \geq 0 \tag{3.23}$$

with

$$1 \leq i \leq n1$$

$$1 \leq j \leq n2$$

where

- $h(x_i)$ is the i-th equality constraint.
- $g(x_i)$ is the j-th inequality constraint.
- n1 is the total number of equality constraints
- n2 is the total number of inequality constraints

The problem is then converted in to an unconstrained optimization problem as follows.

$$\Psi = E + \Re\Big[\sum_{i=1}^{n1} h_i^2 + \phi\Big] \tag{3.24}$$

where

- $\Re$ is the penalty parameter and its value is taken as 10000 in the present implementation.
- h_i is the i-th equality constraint value.
- $n1$ is the total number of equality constraints.
- $\phi = \sum_{j=1}^{n2} \begin{cases} g_j^2 & : \quad g_j < 0 \\ 0 & : \quad g_j \geq 0 \end{cases}$ where $n2$ is the total number of inequality constraints.

The fitness function is a decreasing function of Ψ. Lower the value of Ψ higher will be the fitness. A crossover probability of 0.91-0.98 and a mutation probability of 0.01-0.15 has been used.

Once the good guess is obtained, the rest is to get the exact solution by using conventional optimization algorithms. For this purpose, the entire code is rewritten in MATLAB. A function called **constr** is available in MATLAB which can be used for constrained minimization.

Chapter 4

The inside mapping

The problem of mapping the entire region (the external boundary and the region inside) was split into two relatively simpler problems.The previous chapter explained the algorithms that have to be implemented for mapping the external boundary. In this chapter, the algorithm which has to be implemented after the external boundary has been mapped is presented. All the assumptions in the previous chapter are still valid in this chapter too.

4.1 The problem of inside mapping

The main aim of this problem is to find the positions of all the nodes which were not considered while mapping the external boundary. Each internal boundary has to be shrunk to some positions inside the new external boundary obtained in the previous chapter. This new external boundary defined by previous chapter will be applied as a set of boundary conditions to the system of equations which will be used to solve the current problem. This problem can be directly solved for the position of inner nodes by finite element method if none of the internal boundaries exist by applying those boundary conditions. If even one internal boundary exist, then the position of the points to which the internal boundaries shrink has to be found by solving a new optimization problem along with some constraints to minimize the energy of all the inner springs and later the position of all the inner nodes can be obtained by solving it by finite element method by applying the boundary conditions.

4.2 The existence of internal boundaries

The positions of the inner nodes are dependent upon the positions of the nodes on the external boundary and on the positions of the nodes to which all the nodes of a particular internal boundary shrinks. If in a case where there are no internal boundaries, the positions of the inner nodes solely depend on the position of the nodes on the external boundary because the system of equations are solvable by mere application of the displacement boundary condition for the nodes on the external boundary, by finite element method, rendering the positions of the inner nodes to be dependent only on the positions of the nodes of the external boundary.

If one or more internal boundaries exist, the positions to which each internal boundary will shrink is not known. Now in this case, the distortion energy can still be varied by arbitrarily changing the positions of the nodes to which the internal boundaries will get shrunk. So the distortion energy of the system is a function of the these unknown positions. This problem is essentially an optimization problem.

4.3 Calculation of forces at all nodes

As already mentioned, every edge is considered as a spring of finite stiffness. The whole system can be considered as a cluster of springs. While mapping the external boundary, the calculation of forces were not needed but only displacements were needed. Here in the inside mapping, nodal forces have to be calculated for all the nodes including those on the external boundary as well. In order to get the nodal forces from displacements, a certain set of equations have to be computed. They can very well be represented in a matrix form as given below.

$$[F] = [G][\Delta] \tag{4.1}$$

where

- F is a column matrix denotes force at each node.
- G is called as **global stiffness matrix** of all the nodes which is a square matrix.
- Δ is the column matrix of nodal displacements.

4.3.1 Assembly of global stiffness matrix

For any edge which is considered as a spring of finite stiffness as shown in the figure 4.1, the stiffness is given by

$$s = AE/l \tag{4.2}$$

where

- A is the area of cross-section.
- E is the Young's modulus of elasticity of the material of the element.
- l is the initial length of the element.

The relationship between force and displacement for this element under equilibrium condition can be obtained as

$$\begin{bmatrix} f_{n1} \\ f_{n2} \end{bmatrix} = s \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} \Delta l_1 \\ \Delta l_2 \end{bmatrix} \tag{4.3}$$

where

- f_{n1} and f_{n2} are the forces at the nodes of the element as shown in the Fig, along the axis of the element itself.
- Δl_1 and Δl_2 are the nodal displacements of the element along the axis of the element itself.

The nodal forces and displacements for the same element in 3-d w.r.t to an universal frame of reference as shown in the fig can be obtained from the previous equations as follows.

Let

$$\begin{bmatrix} \zeta \end{bmatrix} = \begin{bmatrix} c_\alpha^2 & c_\alpha c_\beta & c_\alpha c_\gamma & -c_\alpha^2 & -c_\alpha c_\beta & -c_\alpha c_\gamma \\ c_\alpha c_\beta & c_\beta^2 & c_\beta c_\gamma & -c_\alpha c_\beta & -c_\beta^2 & -c_\beta c_\gamma \\ c_\alpha c_\gamma & c_\beta c_\gamma & c_\gamma^2 & -c_\alpha c_\gamma & -c_\beta c_\gamma & -c_\gamma^2 \\ -c_\alpha^2 & -c_\alpha c_\beta & -c_\alpha c_\gamma & c_\alpha^2 & c_\alpha c_\beta & c_\alpha c_\gamma \\ -c_\alpha c_\beta & -c\beta^2 & -c_\beta c_\gamma & c_\alpha c_\beta & c_\beta^2 & c_\beta c_\gamma \\ -c_\alpha c_\gamma & -c_\beta c_\gamma & -c_\gamma^2 & c_\alpha c_\gamma & c_\beta c_\gamma & c_\gamma^2 \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta y_1 \\ \Delta z_1 \\ \Delta x_2 \\ \Delta y_2 \\ \Delta z_2 \end{bmatrix} \tag{4.4}$$

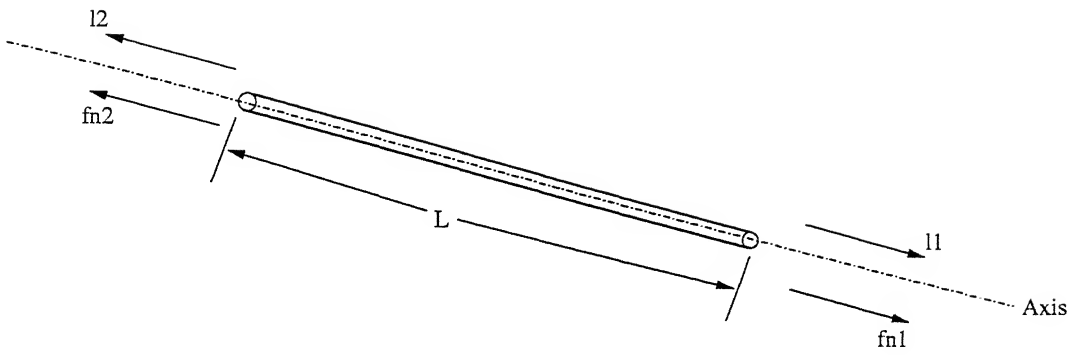


Figure 4.1: An edge assumed as an element having finite stiffness

then

$$\begin{bmatrix} f1_x \\ f1_y \\ f1_z \\ f2_x \\ f2_y \\ f2_z \end{bmatrix} = s \begin{bmatrix} \zeta \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta y_1 \\ \Delta z_1 \\ \Delta x_2 \\ \Delta y_2 \\ \Delta z_2 \end{bmatrix} \tag{4.5}$$

where

- The left side column matrix is the nodal force matrix for the element shown.
- The matrix on the right side is the nodal displacement matrix.
- The square matrix ζ containing the cosine terms is the orthogonal transformation matrix. c_α, c_β,c_γ, are the direction cosines ($\cos\alpha$, $\cos\beta$,$\cos\gamma$) of the line segment joining the two nodes of the element taken in to consideration w.r.t to the global frame of reference as shown in the figure 4.2.
- s is the stiffness of the element.

In the same way, the nodal forces and displacements for the same element in 2-d w.r.t an universal frame of reference as shown in the figure 4.3 can be obtained as follows.

Let

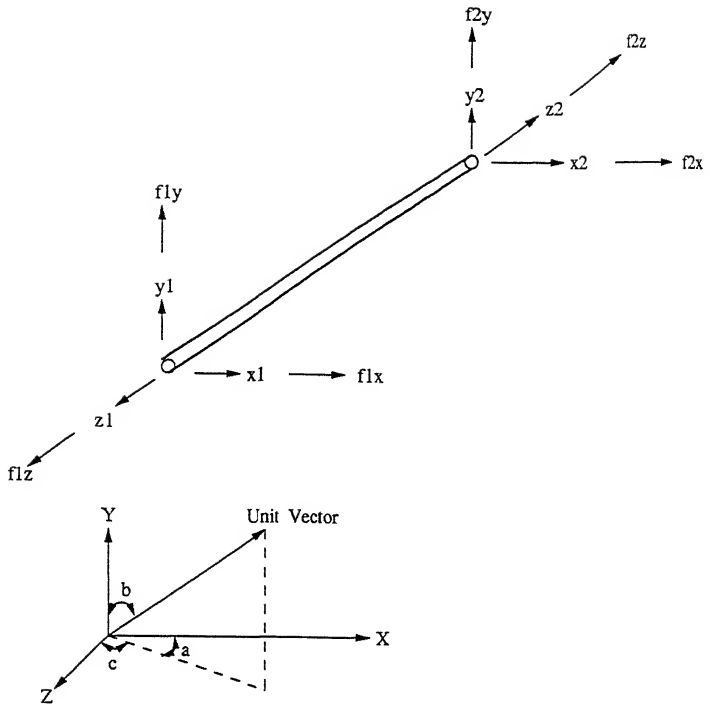


Figure 4.2: Forces and displacements of an element in 3-d

$$\left[\ \xi\ \right] = \begin{bmatrix} c_\theta^2 & c_\theta s_\theta & -c_\theta^2 & -c_\theta s_\theta \\ c_\theta s_\theta & s_\theta^2 & -c_\theta s_\theta & -s_\theta^2 \\ -c_\theta^2 & -c_\theta s_\theta & c_\theta^2 & c_\theta s_\theta \\ -c_\theta s_\theta & -s_\theta^2 & c_\theta s_\theta & s_\theta^2 \end{bmatrix} \tag{4.6}$$

then

$$\begin{bmatrix} f1_x \\ f1_y \\ f2_x \\ f2_y \end{bmatrix} = s\left[\ \xi\ \right] \begin{bmatrix} \Delta x_1 \\ \Delta y_1 \\ \Delta x_2 \\ \Delta y_2 \end{bmatrix} \tag{4.7}$$

where

- The left side column matrix is the nodal force matrix for the element shown.
- The matrix on the right side is the nodal displacement matrix.
- The square matrix ξ containing the trigonometric terms is the orthogonal transformation matrix. c_θ, s_θ, are the $\cos\theta$ and $\sin\theta$ respectively and θ is the

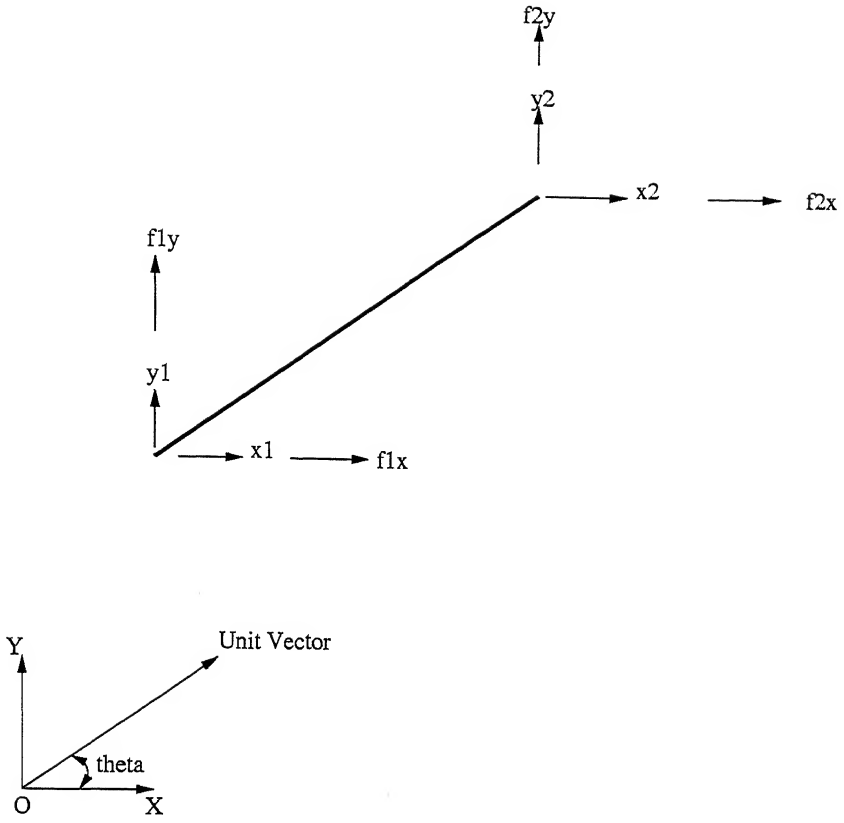


Figure 4.3: Forces and displacements of an element in 2-d

angle measured anticlockwise from X-axis for the line segment joining the two nodes of the element taken in to consideration w.r.t to the global frame of reference as shown in the figure 4.3.

- s is the stiffness of that particular element.

Let $\left[\ k1\ \right] = s\left[\ \zeta\ \right]$ and $\left[\ k2\ \right] = s\left[\ \xi\ \right]$.

This new matrices $K1$ and $K2$ can be called as element stiffness matrices. Every element in the polyhedron or polygon, also referred to as spring or edge will give out an element stiffness matrix of its own. As noticed above, the size of each element stiffness matrix will be 6×6 in 3-d or 4×4 in 2-d. When all these element stiffness matrices are assembled corresponding to each node and element in the entire system,

a square matrix of size 3n×3n in 3-d or 2n×2n in 2-d will be obtained finally where n is the total number of nodes all over the polyhedron or polygon.

In 3-d, the global stiffness matrix will be of the form

$$\left[\ G_{3d}\ \right] = \begin{bmatrix} g_{11} & g_{12} & \cdots & g_{1(3n)} \\ g_{21} & g_{22} & \cdots & g_{2(3n)} \\ \vdots & \vdots & \ddots & \vdots \\ g_{(3n)1} & g_{(3n)2} & \cdots & g_{(3n)(3n)} \end{bmatrix} \tag{4.8}$$

In 2-d the global stiffness matrix will be of the form

$$\left[\ G_{2d}\ \right] = \begin{bmatrix} g_{11} & g_{12} & \cdots & g_{1(2n)} \\ g_{21} & g_{22} & \cdots & g_{2(2n)} \\ \vdots & \vdots & \ddots & \vdots \\ g_{(2n)1} & g_{(2n)2} & \cdots & g_{(2n)(2n)} \end{bmatrix} \tag{4.9}$$

The equilibrium equations of entire system in 3-d can now be rewritten as

$$\begin{bmatrix} f1_x \\ f1_y \\ f1_z \\ \vdots \\ fn_x \\ fn_y \\ fn_z \end{bmatrix} = \left[\ G_{3d}\ \right] \begin{bmatrix} \Delta x_1 \\ \Delta y_1 \\ \Delta z_1 \\ \vdots \\ \Delta x_n \\ \Delta y_n \\ \Delta z_n \end{bmatrix} \tag{4.10}$$

where

- The matrix on the L.H.S is the nodal force matrix.
- The matrix on the R.H.S is the nodal displacement matrix.

Similarly, the equilibrium equations of the entire system in 2-d can be rewritten as

$$\begin{bmatrix} f1_x \\ f1_y \\ \vdots \\ fn_x \\ fn_y \end{bmatrix} = \left[\ G_{2d}\ \right] \begin{bmatrix} \Delta x_1 \\ \Delta y_1 \\ \vdots \\ \Delta x_n \\ \Delta y_n \end{bmatrix} \tag{4.11}$$

where

- The matrix on the L.H.S is the nodal force matrix.
- The matrix on the R.H.S is the nodal displacement matrix.

4.4 Solution to the problem with no internal boundaries

As discussed earlier, if there are no internal boundaries, mere substitution of the boundary conditions to the equilibrium equation obtained above will yield us the position of the inner nodes. The vector difference between a point in the initial polyhedron or polygon and the corresponding point on the mapped polyhedron or polygon will give the displacement boundary condition for a node which lies on the external boundary. After applying boundary conditions, the equilibrium equations get modified. Then the modified equations are solved by MATLAB. Nodal displacements are nothing but incremental displacements. Algebraically adding each nodal displacement to each inner node will yield us the position of the inner nodes w.r.t. to the universal frame of reference considered.

4.5 Solution to the problem with one or more internal boundaries

The position of inner nodes in the mapped polyhedron or polygon, is a function of the position of the nodes on the external boundary and the positions to which the internal boundaries will eventually shrink to a point inside the polyhedron or polygon. The existence of internal boundaries yield another set of independent variables, which are the positions to which the internal boundaries will shrink. Different positions when arbitrarily chosen will give different distortion energy. Hence once again the distortion energy has to be minimized to get the value of the independent variables.

4.5.1 The objective function

The objective function is the distortion energy of all edges which are considered as elements with finite stiffness which lie inside the polyhedron. For this purpose a

linked list has to be defined by extracting all the edges which are inside.

The algorithm to extract all the edges which are inside the polyhedron is explained in the appendix (algorithm 13).

Extraction of edges which are inside the polygon in 2-d

A linked list *inneredges* of type **edges** is created by extracting all the edges which are inside the polygon in 2-d by using the same idea of the previous algorithm.

The objective function, the distortion energy of the edges which are inside is given by

$$\mathcal{E} = \frac{1}{2}\sum_{i=1}^{N} k_i(\Delta l)_i^2 \tag{4.12}$$

where

- k_i is the stiffness of the i-th edge in the linked list *inneredges*.
- Δl is the deformation of the i-th edge in the linked list *inneredges*.
- N is the total number of edges inside the polyhedron or polygon.

4.6 The constraints

In this inside mapping too, constraints have to be applied. As we are going to use GA, the constraints are fine tuned to best represent the purpose of its application.

The positions of the inner nodes and the positions of the points to which the internal boundaries shrink will be considered as unknowns. This is done due to numerical exception problem which is explained in detail in section 4.6.2. The total number of such variables will be 3*(n1+n2) in 3-d or 2*(n1+n2) in 2-d, where n1 is the total number internal boundaries and n2 is the total number of inner nodes. So the initial population in GA is also made to have 3*(n1+n2) in 3-d or 2*(n1+n2) members.

All the nodes of a particular internal boundary will be directly allotted this initialized values. This means that, all the nodes on this internal boundary has

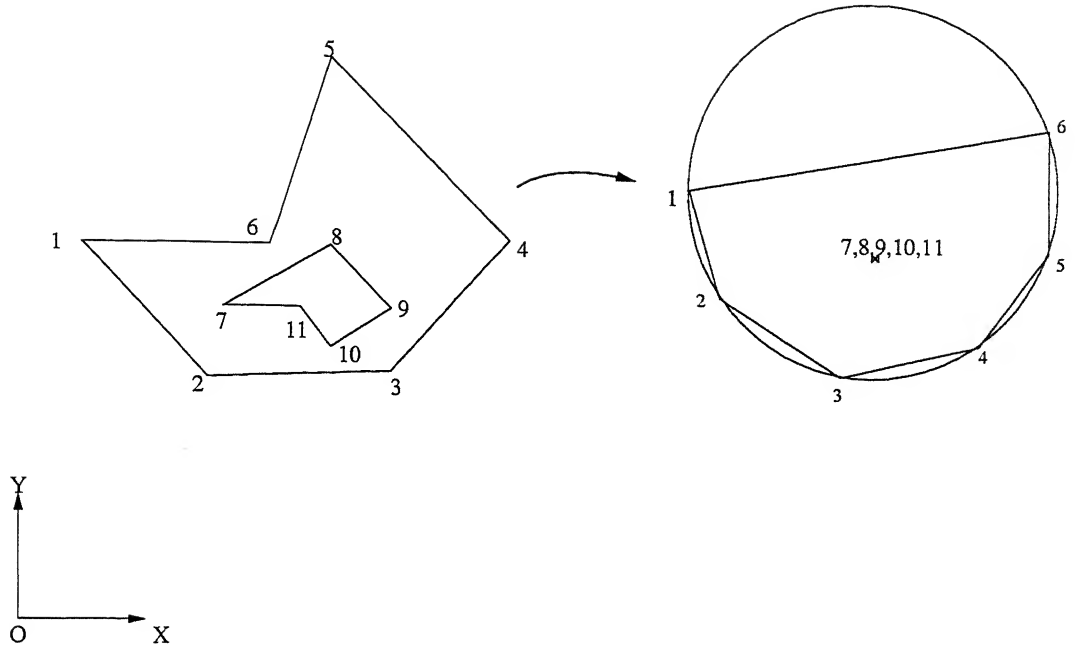


Figure 4.4: Internal boundary shrunk to a point inside the polygon

virtually converged to this point, which was initialized by GA. The edges on the internal boundaries are assumed to have been compressed to their full length as shown in the figure 4.4.

4.6.1 Inequality constraints in 3-d

Constraints to ensure that the nodes to which the internal boundaries are shrunk separately, should lie well inside the external boundary

We had already defined the semantics of the external boundary well. If the normal is computed according to the convention assumed, it will point outwards from the polyhedron for each triangle on the external boundary.

This is an inequality type constraint and it is applied as follows.

- **FOR LOOP BEGIN** from $i = 1$ to *totalinternalboundaries*

- Take out the point to which the i^{th} internal boundary has shrunk. This is the value provided by GA.
- $j = 1$
- **DO**
  * Find out the vector from the point taken above to any one of the nodes of the triangle *extbound*(j). Let this be $\vec{A}$.
  * Find out the normal of the triangle *extbound*(j). Let this be $\vec{N}$.
  * The constraint is modelled as follows.

$$g_k = \vec{A}.\vec{N} \geq 0 \tag{4.13}$$

    The above constraint is of type greater than zero.
  * $j = j + 1$
- **WHILE LOOP** until $j \leq j_{last}$

- **FOR LOOP END**

where

- j_{last} is the total triangles on the external boundary.

The total number of such constraints will be *totalinternalboundaries* times total number of triangles on the external boundary.

4.6.2 Inequality constraints in 2-d

Constraints to ensure that the nodes to which the internal boundaries are shrunk separately should lie well inside the external boundary

The semantics of the polygon on 2-d was well defined by ordering the numbering of the external boundary in anticlockwise direction. If the signed area of the triangle formed by two nodes of any edge taken in the same order as they are stored in the linked list and an arbitrary point, is positive, then that point chosen arbitrarily is inside the polygon. If in the case it is negative, then that point is outside. If the

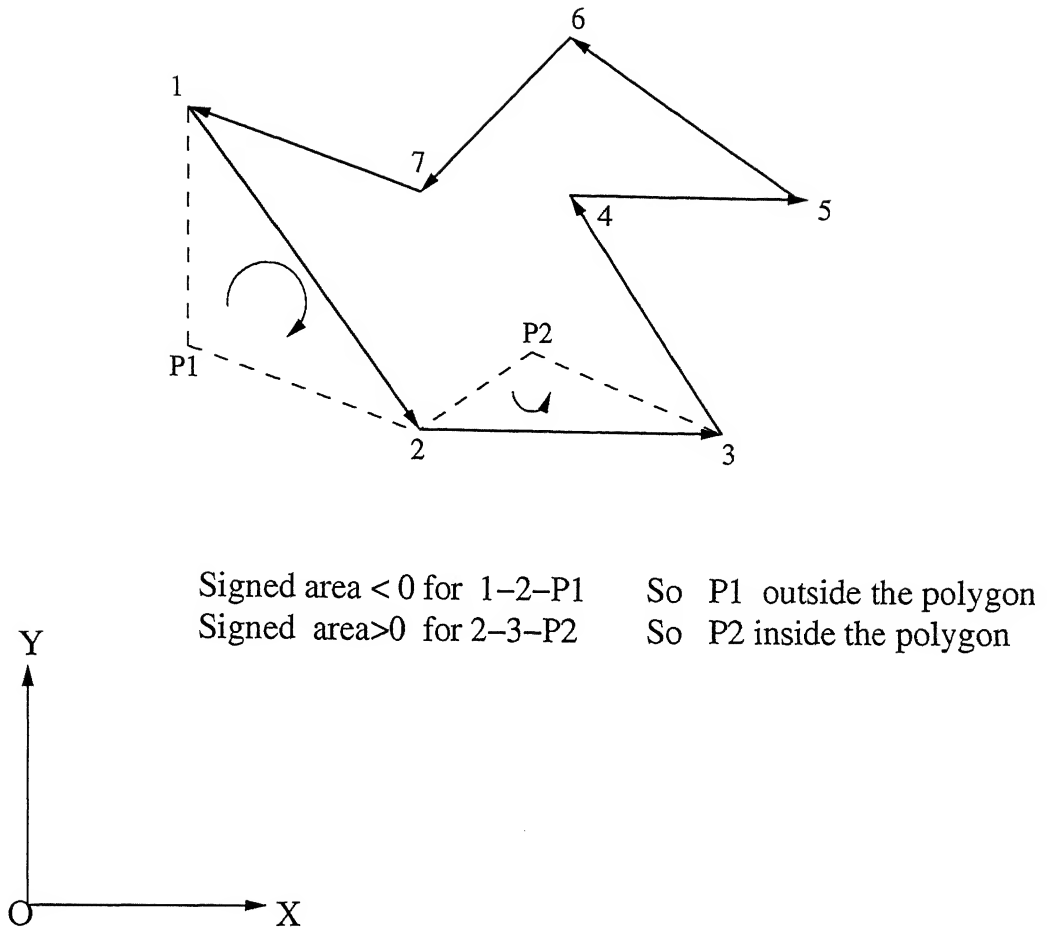


Figure 4.5: Point membership classification of a point with a polygon

area is zero, then that point lies on the polygon. These cases are shown in the figure 4.5. As it is observed that the value should be positive, this is an inequality type of constraint.

The following are the steps to model this constraint.

- **FOR LOOP BEGIN** from $i = 1$ to *totalinternalboundaries*
  - Take the point to which the i-th internal boundary has shrunk. Let it be P.
  - $j = 1$

- **DO**
  - Take out the nodes on the edge stored in *extbound(j)* in the same order. Let them be *N1* and *N2*.
  - Find out the signed area of the triangle taken in the order *N1,N2,P*. Let this be A.
  - The constraint is modelled as follows.

$$g_k = A - \epsilon \geq 0 \tag{4.14}$$

    where
    - ϵ is the lower limit on the area so as to make area not to go lesser than this limit.

    The above constraint is of type greater than or equal to zero.
  - $j = j + 1$
- **WHILE LOOP** until $j \leq j_{last}$

- **FOR LOOP END**

4.6.3 Equality constraints in 3-d or in 2-d

1. **Constraints to impose equilibrium**

   The positions of the nodes inside the polyhedron or polygon are dependent upon the nodes on the external boundary and on the nodes to which each internal boundary will separately shrink. If once the position of the nodes required to find out the position of the nodes inside the polyhedron or polygon are found out, the rest is to apply this boundary condition to the same equilibrium equation obtained to solve for the position of the inner nodes. But, if during one of the iterations of this optimization problem, if some values assigned to the position of the nodes, to which the internal boundaries will shrink separately, are infeasible i.e. if that node lies on or outside the polyhedron or polygon, a run time numerical exception will be created as the matrix obtained after applying boundary conditions from the global stiffness matrix will still be non-invertible. So, the problem is solved directly instead of inverse methodologies.

For i-th node, this constraint in 3-d is modelled as follows

$$\begin{bmatrix} h_i \\ h_{i+1} \\ h_{i+2} \end{bmatrix} = \begin{bmatrix} g_{(3i-2)1} & g_{(3i-2)2} & \cdots & g_{(3i-2)3n} \\ g_{(3i-1)1} & g_{(3i-1)2} & \cdots & g_{(3i-1)3n} \\ g_{(3i)1} & g_{(3i)2} & \cdots & g_{(3i)3n} \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta y_1 \\ \Delta z_1 \\ \vdots \\ \Delta x_n \\ \Delta y_n \\ \Delta z_n \end{bmatrix}$$

$$= \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{4.15}$$

This constraint is applied in 2-d as follows

$$\begin{bmatrix} h_i \\ h_{i+1} \end{bmatrix} = \begin{bmatrix} g_{(2i-1)1} & g_{(2i-1)2} & \cdots & g_{(2i-1)2n} \\ g_{(2i)1} & g_{(2i)2} & \cdots & g_{(2i)2n} \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta y_1 \\ \vdots \\ \Delta x_n \\ \Delta y_n \end{bmatrix}$$

$$= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{4.16}$$

where n is the total number of nodes on the entire polyhedron. There will be 2N equality constraints, if there are N inner nodes.

Chapter 5

Planning the path

The initial region is now mapped to a target convex region on to a sphere by minimally distorting it. The data of this mapped region exists in the form of mapped tetrahedrons in 3-d or triangles in 2-d. As the connectivities of all the tetrahedrons or triangles are preserved, the correspondence between the original and the mapped tetrahedron or triangle is deemed to be existing.

For each tetrahedron or triangle, the co-ordinates of the initial and final position of its nodes are known. We now need to know in general any arbitrary point in or on the tetrahedron where it would be mapped in the mapped tetrahedron. For this purpose, linear mapping is assumed.

5.1 Linear mapping

If a linear correlation has to be obtained from a tetrahedron defined in η,ξ,ψ coordinate system to its distorted form represented in another coordinate system say x,y,z (figure 5.1) then, the following linear mapping principle is assumed.

$$x = C_1 + C_2\eta + C_3\xi + C_4\psi \tag{5.1}$$

$$y = C_5 + C_6\eta + C_7\xi + C_8\psi \tag{5.2}$$

$$z = C_9 + C_{10}\eta + C_{11}\xi + C_{12}\psi \tag{5.3}$$

where

- x,y,z are the coordinates of the point in the mapped space corresponding to a point η, ξ, ψ in the original space as shown in the figure 5.1.

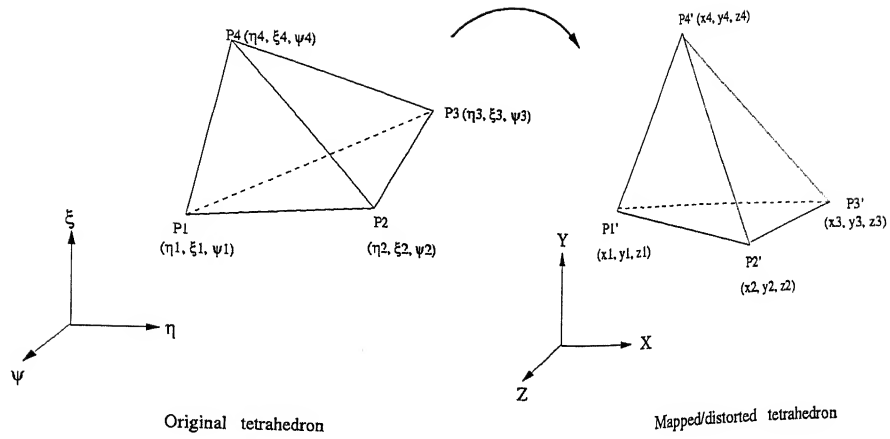


Figure 5.1: Initial and distorted tetrahedrons

- C_1 to C_{12} are constants.

The above states that any co-ordinate in the mapped configuration is a linear function of the co-ordinates in the original configuration. In the above equations, there are 12 constants. For each node of a tetrahedron, there will be 3 equations and totally there would be 12 equations for all the 4 nodes. Those equations can be solved for the constants to obtain the co-ordinate of any arbitrary point in or on the tetrahedron in the mapped configuration.

In the case of 2-d, the equations for linear mapping are

$$x = C_1 + C_2\eta + C_3\xi \tag{5.4}$$

$$y = C_4 + C_5\eta + C_6\xi \tag{5.5}$$

where

- x,y are the coordinates of the point in the mapped space corresponding to a point η, ξ in the original space as shown in the figure 5.2.

- C_1 to C_6 are constants.

For a triangle in 2-d as shown in the figure 5.2, 6 equations in 6 unknowns will be obtained.

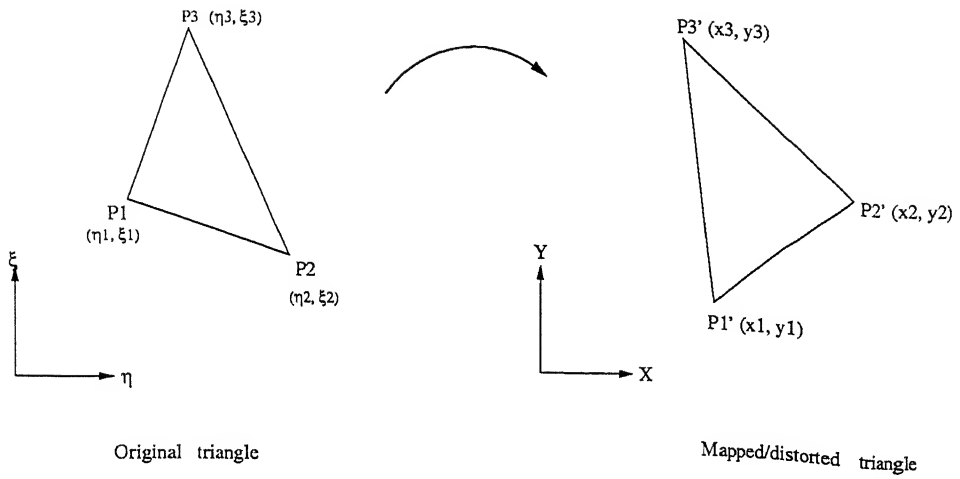


Figure 5.2: Initial and distorted triangles

5.2 Finding the path

The corresponding co-ordinates of the initial and goal configuration are obtained in the mapped region from the original region by linear mapping. These two points are joined by a straight line as the mapped region is convex. This line joining the initial and the final configurations if inverted back to the original region is the required path.

In order to invert the straight line back to the original region, it is divided into 100 segments. For each point on the line segments, the tetrahedron on which that point lies should be found and those points are inverted back to the original region by linear mapping. The required path is the curve connecting all such points in the original region.

In this thesis, to find the path, the algorithm to plan the path defined in [19] is used.

Chapter 6

Results and discussions

All the algorithms defined in the previous chapters were implemented in a C++ program. Genetic Algorithms(GAs) was used to solve the problem of boundary mapping and inside mapping. Once the above mappings are achieved, the path is planned as explained in chapter 5 by using linear mapping technique. The algorithms defined in the previous chapters are, in principle, capable of planning paths irrespective of whether the regions are simply connected or multiply connected. In this chapter the inferences of the results, results and related discussions are presented.

6.1 The output from GA for boundary mapping and its study

Two problems were considered in 2-d and one in 3-d to plan the path. All the considered cases are multiply connected regions. Figures 6.1 and 6.2 shows the problem in 2-d and 6.3 shows the problem in 3-d. The result from GA for boundary mapping for 2-d for the problem 1 shown in the figure 6.1 is presented in the figures from 6.4 to 6.7. The characteristics of the output from GA for problem 2 in 2-d was almost similar with that of the results obtained for problem 1.

6.1.1 The problems

- The Problem 1 in 2-d is shown in the figure 6.1. This has a single internal boundary which has to be shrunk to a point.

- The Problem 2 in 2-d is shown in the figure 6.2. This has two internal boundaries which have to be shrunk to 2 points.

- The Problem 3 is a 3-d problem and is shown in the figure 6.3. This also has 2 internal boundaries.

In all the problems considered above, the nature of discretization is coarse which makes number of nodes in the polygon or polyhedron less. The path has to be planned between the initial and final configurations which is shown as a point in both the problems. The results for boundary mapping by GA for problem no.1 is shown in the figures 6.4 to 6.7 with increasing value of mutation probability. By increasing value of the mutation probability, a slight improvement in the guess was noted.

The sole aim of using GAs was to get a good search so that it can be used as the good guess in the conventional optimization problem to obtain exact results. But GA gave out a solution whose characteristics were far away from that of a good guess.

The other major problem was the handling of the constraints. Some of the constraints dictated more than others and as a consequence, the left out constraints had no role to play except making the solution highly infeasible. This was not improved by applying high penalty rather it made the solution obtained from GA to cling to the other corner of the infeasibility. In order to remove this problem, all the constraints were normalized, thereby making all the numerical value of all the constraints to be between zero and one. The sole aim of the normalization was to reduce the additional influence of some constraints and at the same time giving equal preference to all the constraints to influence the solution. The main intention to use GA was to avoid a blunt search from being used by the conventional optimization problem. A solution from the GA is assumed to be infeasible but still it is also assumed that it would be a valid search direction. Also, the output from the GA should not make the conventional optimization problem to iterate indefinitely.

As discussed earlier in the chapter 3, there are two types of constraints, constraints of type say A (equations 3.17 for 3-d or equations 3.18 for 2-d) and other of type B (equations 3.15 for 3-d or equations 3.16 for 2-d). to enforce the resulting polyhedron or polygon to be convex which has to be applied simultaneously. The

results obtained (as shown in the figures 6.4 to 6.7), convey that the constraints of type A has a greater influence than that of the constraints of type B.

Beyond certain values of crossover, mutation, population size and number of generations, the guess obtained failed to improve. This forced that guess to be used as the initial guess to the conventional optimization problem.

If we have a close look to the results obtained from the GA (figures 6.4 to 6.7), for boundary mapping, it can be seen that the resulting polygon can be roughly approximated as an highly irregular discrete spiral.

As explained in chapter 3, there are n number of constraints of type A where n is the total combinations denoted by *combinations*. There is only one constraint in type B which is inverse trigonometric. GA gave out a solution which satisfied almost all of the constraints of type A and does not satisfy the constraint of type B due to the former's additional influence on the solution. More specifically, if the constraint of type B is removed, the solution yielded by GA would be close to a star type polygon which is schematically represented with 5 nodes as shown in the figure 6.8. All the constraints of type A are satisfied for the above case.

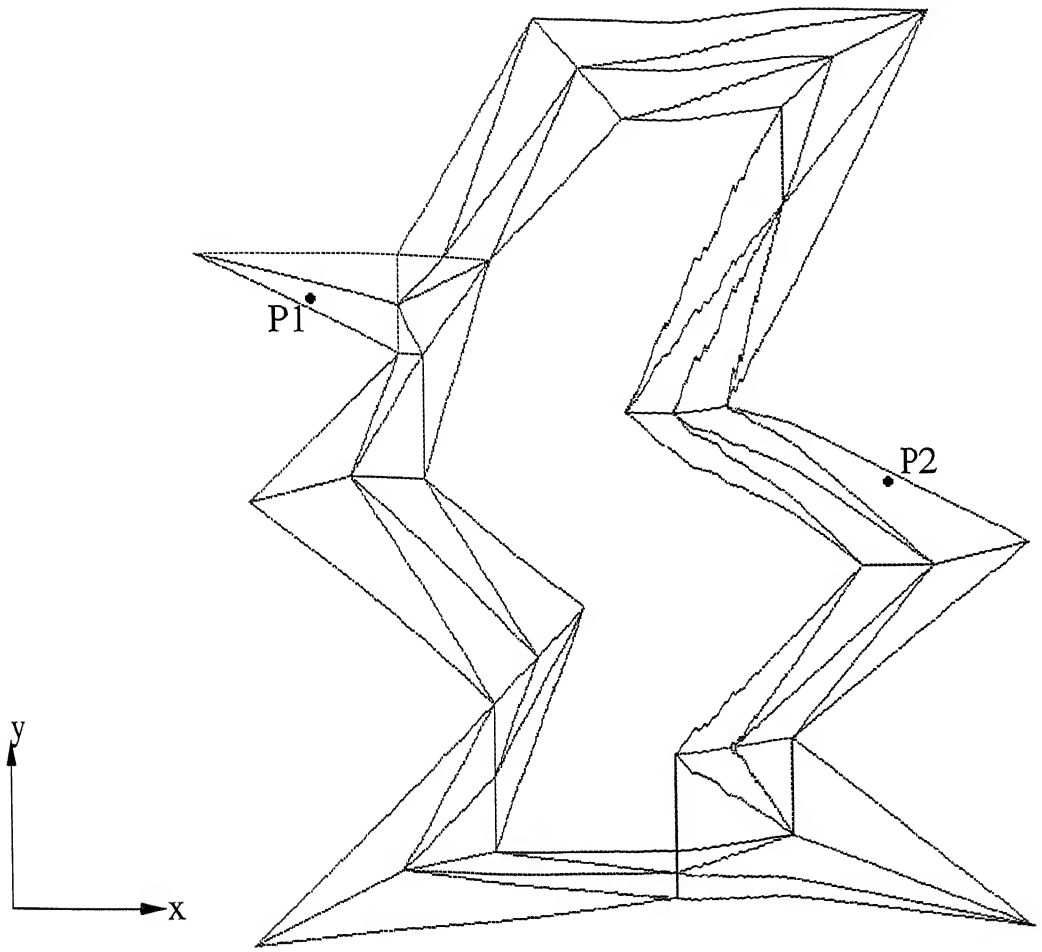
For a problem having large number of nodes on the external boundary, and moreover if it is a 3-d problem, the total number of combinations of two-two triangles and hence the constraints of type A, also increases. Constraint of type B still remains one and its influence on the solution is still diluted and stands neglected. The result for boundary mapping by GA for the 3-d problem considered is shown in the figure 6.9.

Increase in the penalty (if in the case, the constraints of type B are violated) or increase in the mutation probability or generation size or population size had indeed positive effects but were not considerable. Figure 6.7 shows a considerable good guess for a 13 noded problem in 2-d with increased mutation probability and penalty parameter.

As a test case, in order to find out the practical difficuilties in case of 3-d, a polyhedron having 2 internal boundaries (obstacles) was considered. Here also the practical difficuilties remain the same. In fact 3-d cases were tried first. The result from GA was very difficuilt to interpret which prompted to consider 2-d regions.

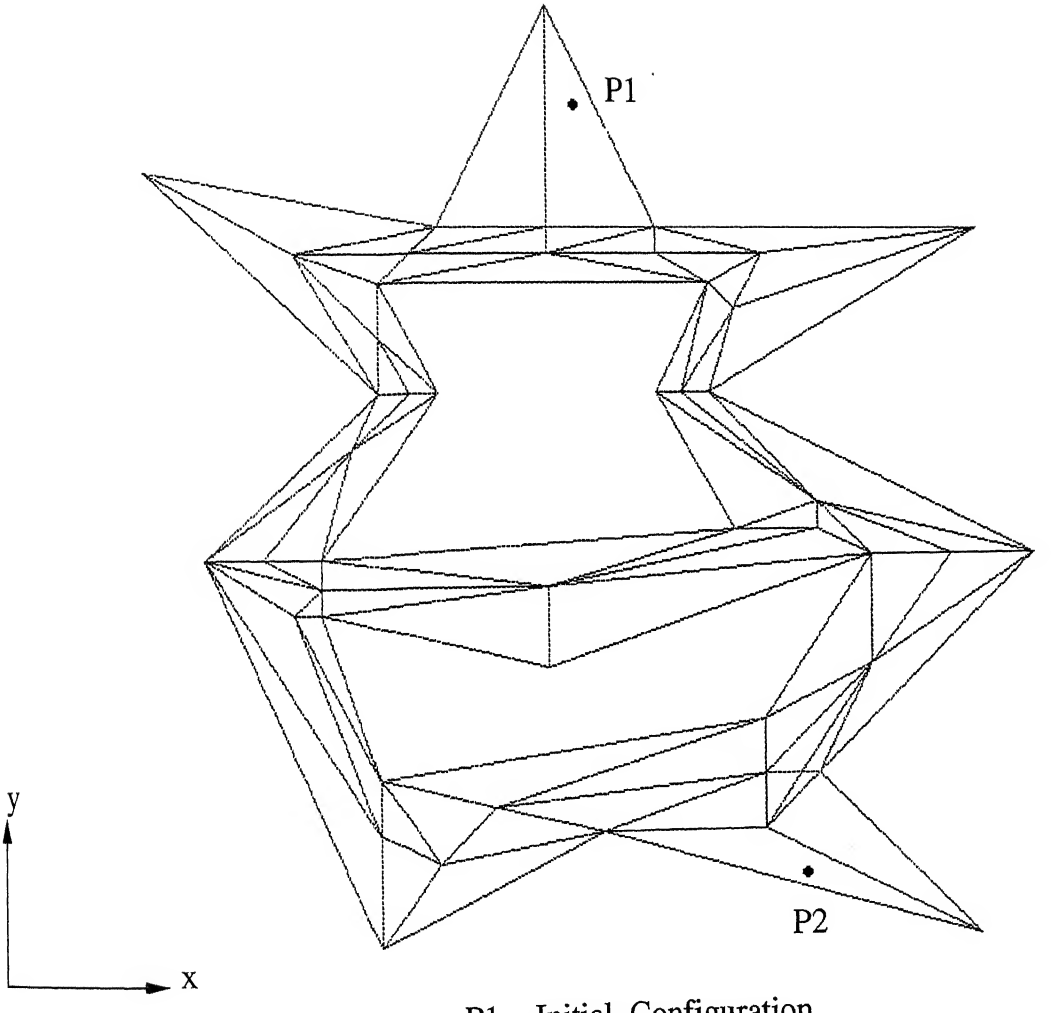
The result for boundary mapping by GA for the 3-d problem considered is shown in the figure 6.9. Here too, it was noted that most of the constraints of type A were

6.12 and 6.13. In order to improve the smoothness of the path, it is mandatory to increase the resolution and hence the total number of nodes in the polygon or polyhedron. This indeed has the negative effect, with the guess coming out to be a bad one. The results were obtained after 1.5 days of computation.



P1 – Initial Configuration
P2 – Final Configuration

Figure 6.1: Problem 1 in 2-d



P1 – Initial Configuration
P2 – Final Configuration

Figure 6.2: Problem 2 in 2-d

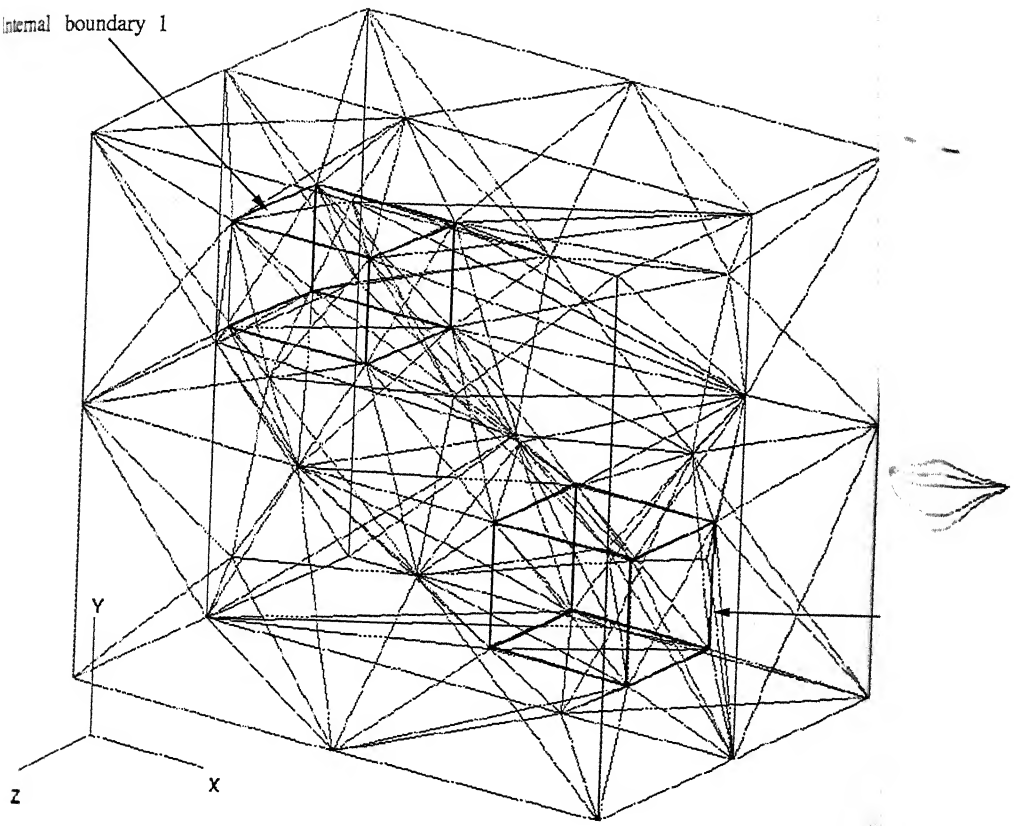
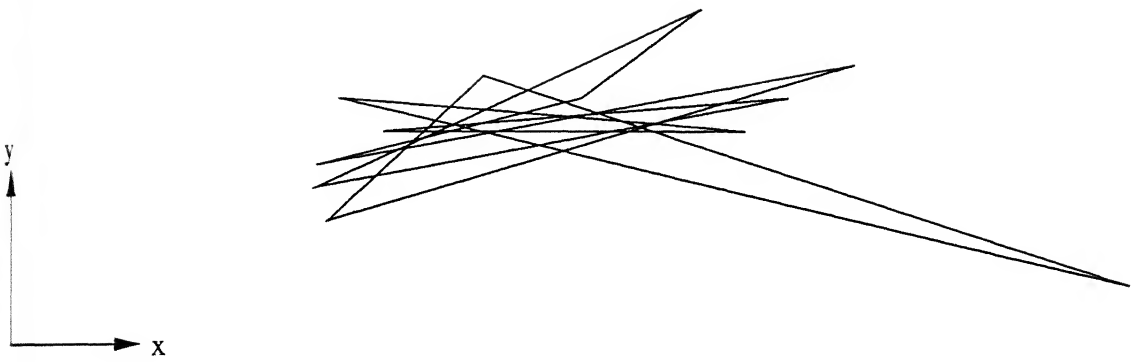
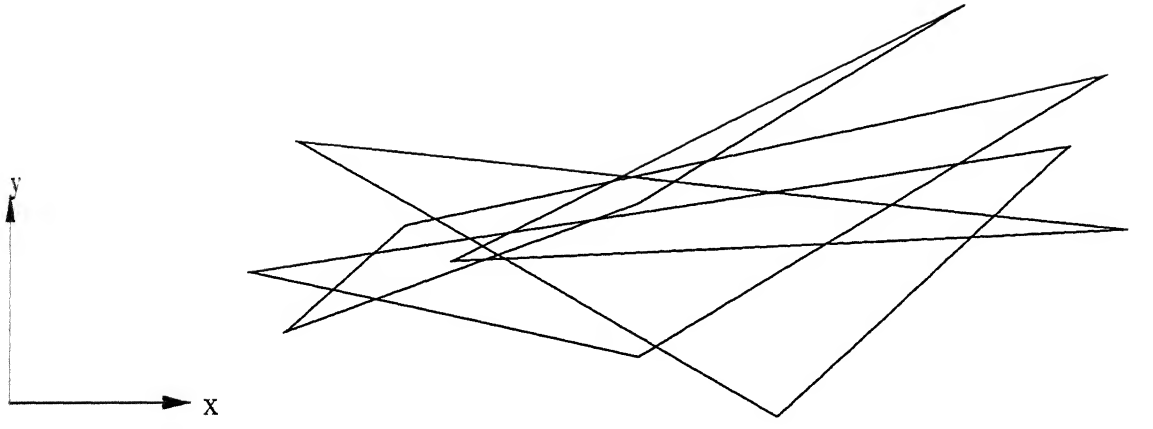


Figure 6.3: 3-d Problem



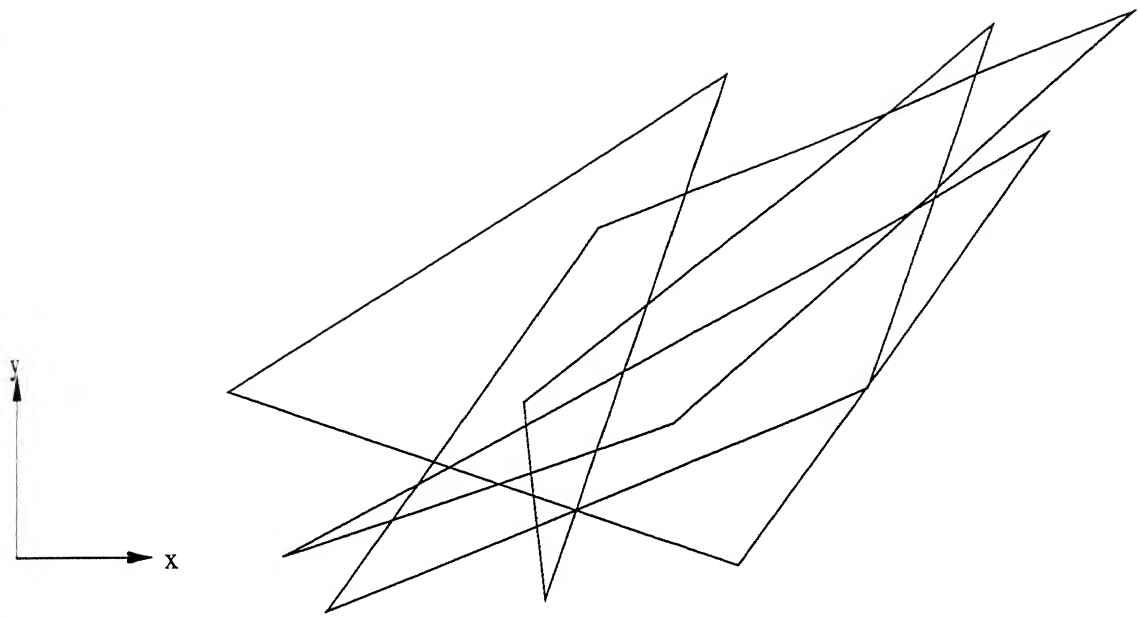
Crossover Probability = 0.98
Mutation Probability = 0.0
Population Size = 500
Total Generations = 1000

Figure 6.4: Result for boundary mapping by GA for problem 1 with mutation probability 0.0



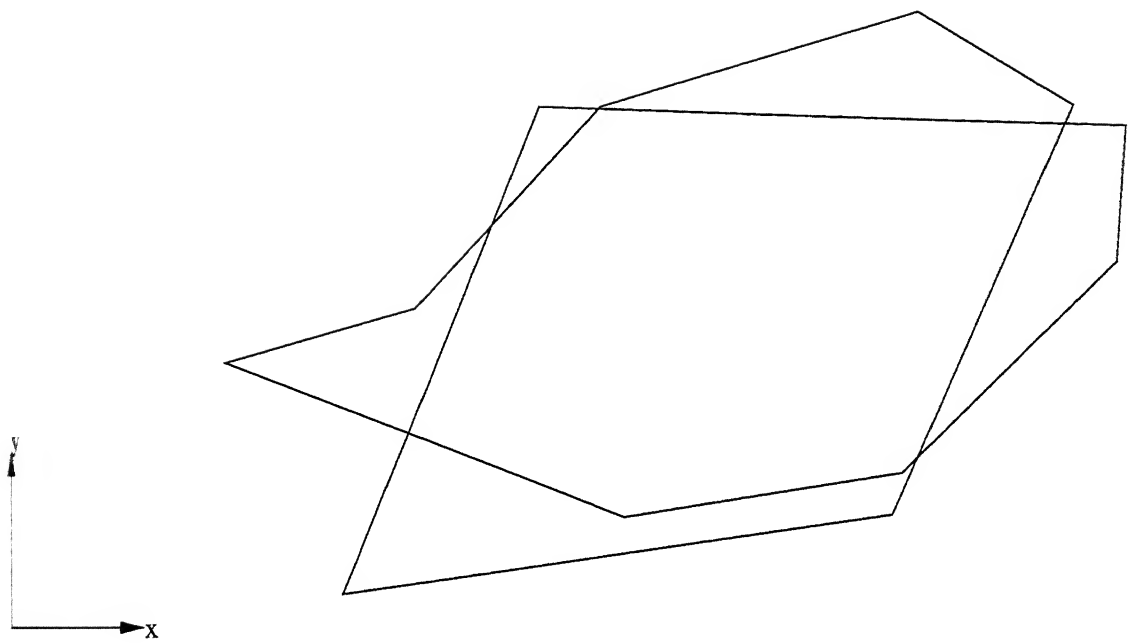
Crossover Probability = 0.97
Mutation Probability = 0.05
Population Size = 600
Total Generations = 1000

Figure 6.5: Result for boundary mapping by GA for problem 1 with mutation probability 0.05



Crossover Probability = 0.95
Mutation Probabilty = 0.098
Population Size = 500
Total Generations = 1000

Figure 6.6: Result for boundary mapping by GA for problem 1 with mutation probability 0.098



Crossover Probability = 0.97
Mutation Probability = 0.11
Population Size = 500
Total Generations = 1000

Figure 6.7: Result for boundary mapping by GA for problem 1 with mutation probability 0.11

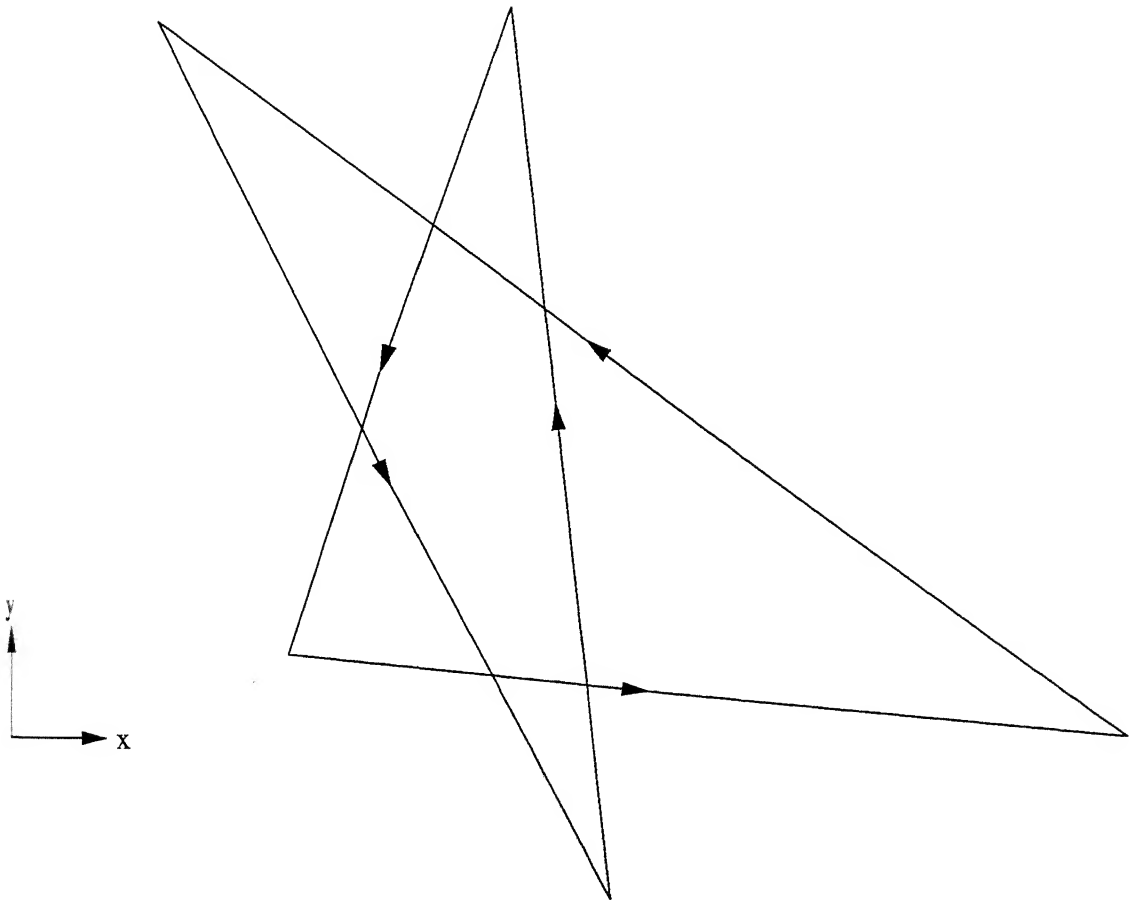
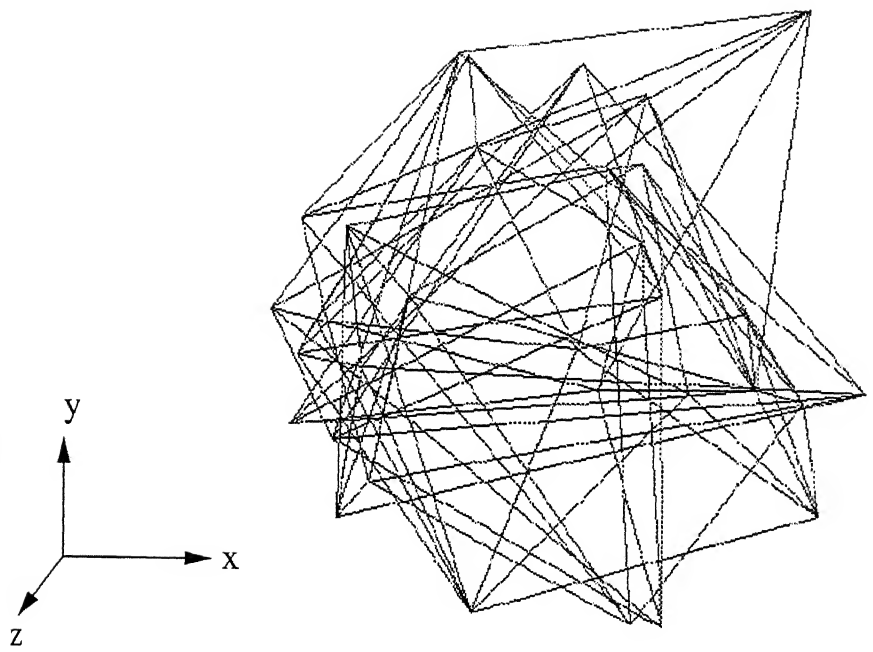


Figure 6.8: A schematic representation of star type polygon violating type B constraint.



Crossover Probability = 0.95
Mutation Probability = 0.08
Population Size = 800
Total Generations = 1000
Total nodes on the external boundary = 32

Figure 6.9: Result for boundary mapping by GA for the 3-d problem showh in the figure 6.3

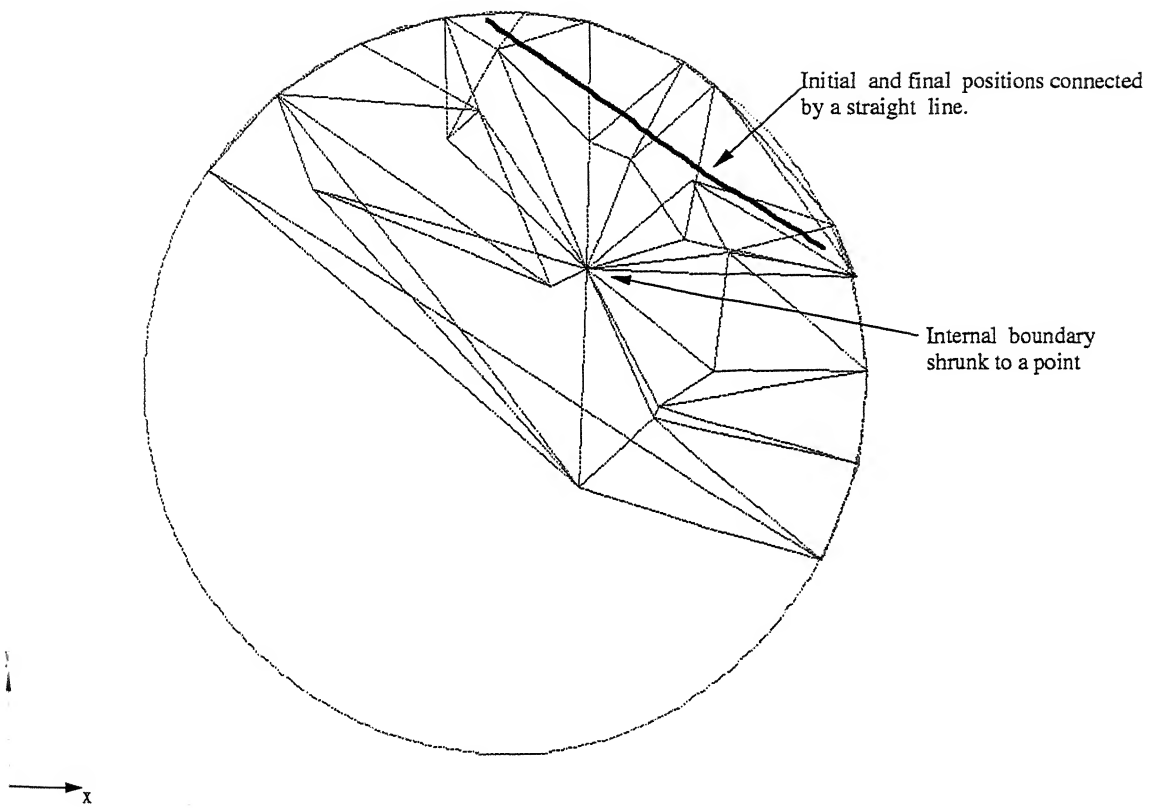


Figure 6.10: The polygon in problem-1 (shown in the figure 6.1) is mapped to a circle with minimum distortion

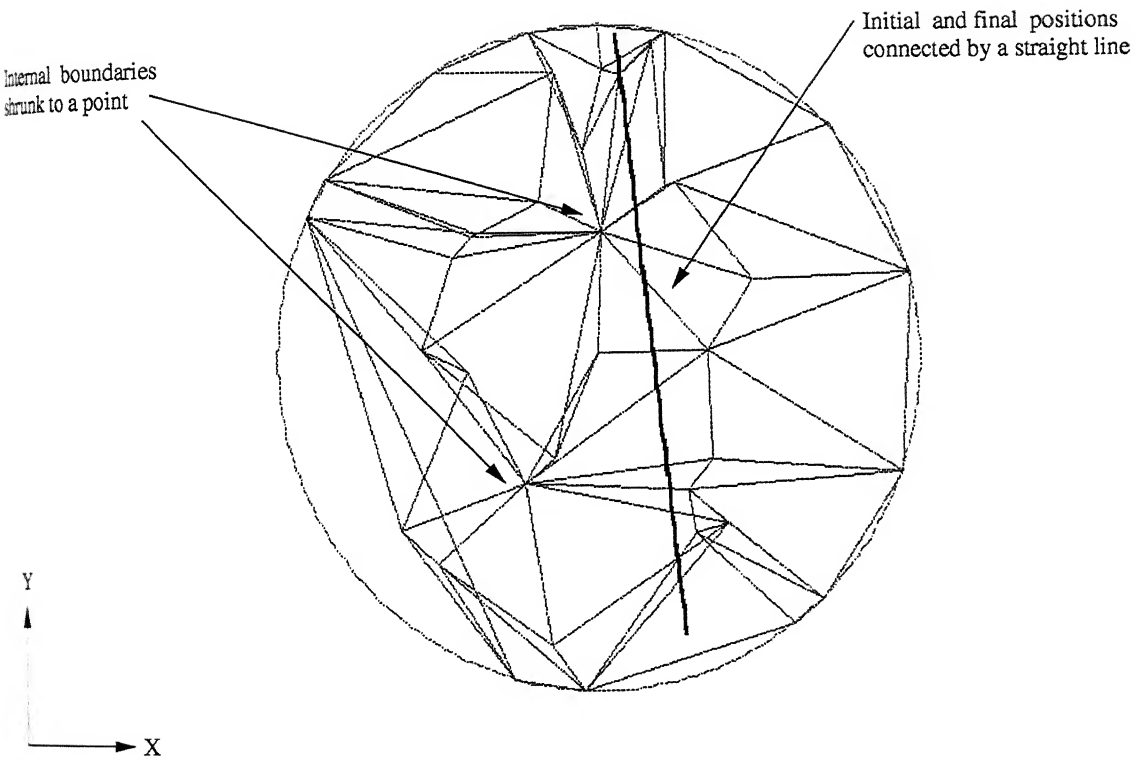


Figure 6.11: The polygon in problem-2 (shown in the figure 6.2) is mapped to a circle with minimum distortion

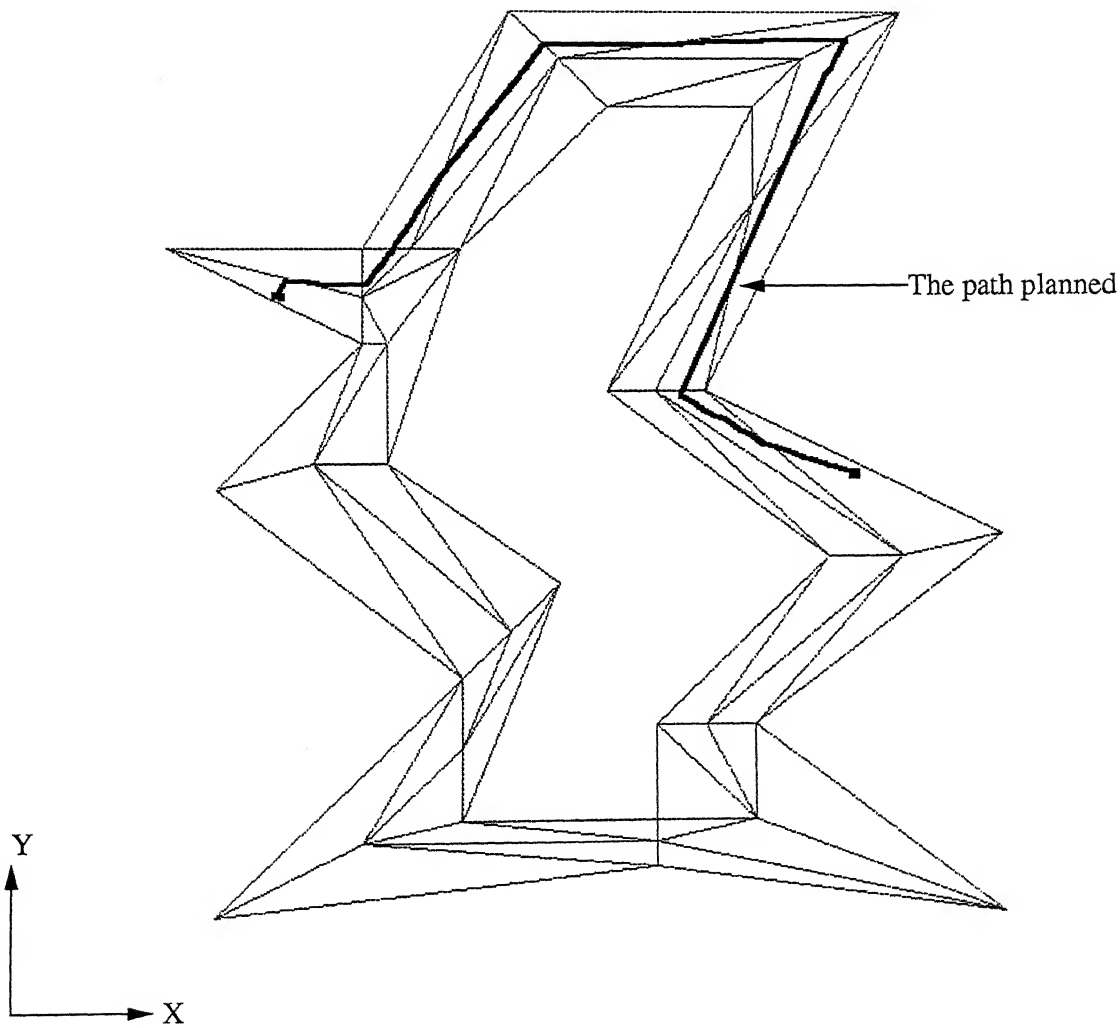


Figure 6.12: Path planned for problem-1 (shown in figure 6.1)

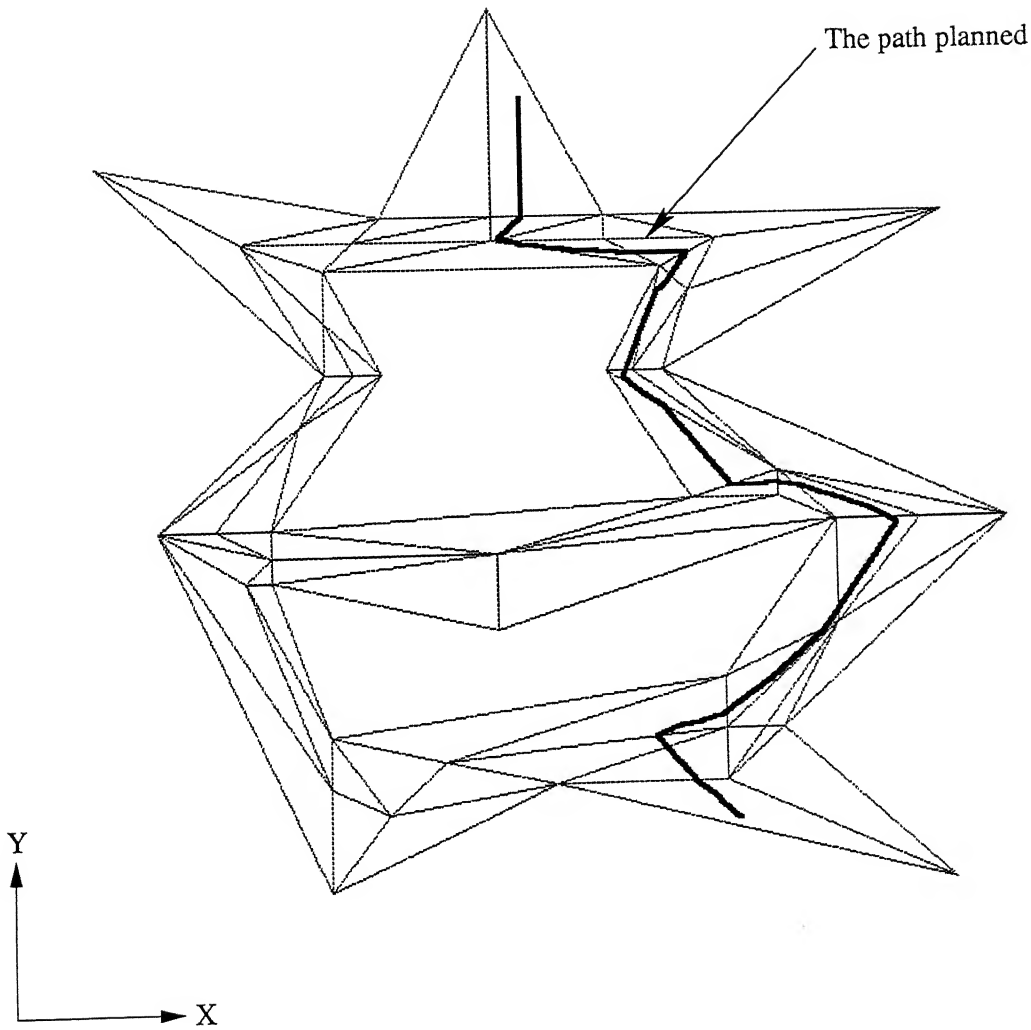


Figure 6.13: Path planned for problem-2 (shown in figure 6.2)

Chapter 7

Conclusions

7.1 Summary

The approach defined and implemented is new of its kind. In chapter 1, some of the optimality criteria were put forth. This method plans the path by mainly taking three of such criteria: maintaing maximum distance from the obstacles, planning a minimum turn path and a path having less distance from the initial configuration to the goal configuration.

In the present work, path planning problem is solved by means of domain mapping with minimum distortion to it, in which, a 3D non-convex region (the C-space) is mapped to a sphere and the C-obstacles are shrunk to points. The rectilinear path is found in the mapped region. The entire path is inverted back to the original region, to get the actual path between two points. The GA part to get a good guess is implemented in a C++ program and the function **constr** in MATLAB is used to solve the optimization problem by using the guess obtained from the previous step. Some of the typical results were presented and discussed.

After studying the intermediate results from GA and final results from MATLAB it can be concluded that,

1. This method can be used for path planning for non-convex regions, whose geometry is equvivalent to that of a sphere. Increase in the total number of nodes makes the problem complicated, which expects an efficient optimization solver to solve. This problem of complexity was considerably felt in this thesis.

2. The path obtained is away from the boundary. In the sense of geometric

length, though the path is not shortest, it is close to the shortest.

3. The path obtained for two 2D problems, explained in chapter 6, is not smooth. It is also distorted at some locations in the region. The path can be smoothened by refining the discretization and again this puts considerable load on the optimization solver. If the optimization solver is much more efficient, a much more complicated region can be dealt with, as complexity is directly proportional to the total number of nodes in the region.

4. Constraint handling poses a severe problem. The total number of constraints is directly proportional to the total number of nodes inside the region.

5. The evaluation of the fitness for GA is computationally expensive due to the presence of several constraints. Eventhough GA is a versatile technique, it is meaningful only if the fitness function evaluation is sufficiently easy to compute.

6. This method proceeds in two phases: a domain mapping phase and a query phase. In the domain mapping phase, two regions are obtained, one is the given region and the other is the mapped region. This mapped region is obtained after running the optimization solver which is time consuming which has to be done only once. The next phase is the query phase, in which, given any two points inside the region, the path is planned. This phase takes very little computation time.

7.2 Future scope

1. **The fitness function:** The fitness of any member should be judged by a fitness function which should ensure mapping steps (shape at each step) to be topologically equivalent to the initial region.

2. **Handling the constraints :** The problems arising out of inefficient handling of the constraints were detailed in chapter 6. Since the complexity of the region solely depends on the total number of nodes representing the region, which is directly proportional to total number of constraints, extreme care

has to be taken in handling the constraints. If a consraint is found to exert less influence on the solution, in addition to normalizing and applying a heavy penalty parameter, some more redundant constraints should be modelled from a different standpoint so that the importance of that particular constraint can be very well re-established to get a very good guess. Note that there should not be any redundant constraints while solving the problem by conventional optimization procedures for exact results as it may create problems.

3. **Design of GA operators :** The operators that are used in GA should be problem specific. In this thesis, a general class of operators were used at first. Later they were slowly modified and some positive effects were noted but they were not considerable. The design of operators are based on the following.

   (a) **The population space :** In this thesis, the population space from which the initial population is obtained is rectangular. The circle or sphere to which the initial region has to be mapped exists inside this rectangular space. If instead of using this rectangular space, if only that circle or sphere is used, it would not only remove the set of constraints given in the equation 3.9 but also reduce the extent of search. Now comes the role of GA operators. These operators have to be re-defined for obtaining new members pertaining to the search space in consideration.

   (b) **Initialization of feasible members :** Instead of arbitrarily initializing some population in which infeasible members also find a position, the initial population is initialized as different random configurations of the mapped region. This can be done by obtaining different randomized solutions by implementing the algorithm proposed in [19]. Note that the path planning algorithm as discussed in that work gives out a feasible solution in our case which is not the optimal one. After initialization of the members, the GA operators should be re-defined again.

   (c) **The nature of search space :** The important step in any optimization problem is the study of the search space that the problem represents. In most of the cases, this is very difficuilt. GA operators should also be designed after a thorough study of this space. A problem specific

conventional optimization algorithm can also be designed if this study is made.

4. **The role of conventional optimization algorithms :** In this thesis the general function for constrained optimization, **constr** in MATLAB is used for getting exact results. A problem specific conventional optimization algorithm shall be used after studying the mathematics of the problem exhaustively.

Appendix A

Description of algorithms

1. ***Algorithm to extract all the triangles from the tetrahedrons***

   Input : Linked list containing all the tetrahedrons called *alltetrahedrons*

   Output : Linked list containing all the triangles from all the tetrahedrons, of type triangles, called *alltriangles*

   Steps :

   (a) $i = 1, j = 1$

   (b) Initiate a linked list of type triangles and be named as *alltriangles*

   (c) **DO**

   (d) Copy the 4 tetrahedrons, which are in *alltetrahedrons(i)* to *alltriangles(j)*

   (e) $j = j + 3$

   (f) $i = i + 1$

   (g) **WHILE LOOP** until $i \leq i_{last}$[1]

   (h) Delete the repeating nodes of the linked list *alltriangles* and make it circular.

[1]The subscript *last* denotes the end of the linked list everywhere.

2. ***Algorithm to extract all the triangles which bound the region either internally or externally***

Input : Linked list *alltriangles*, **Linked list** *alltetrahedrons*

Output : Linked list of type triangles called *boundarytriangles*

Steps :

(a) $i = 1$, $k = 1$,

(b) **DO**

(c) Get the connectivities of triangle *alltriangles(i)* from the linked list *alltriangles*

(d) $j = 1$, $count = 0$

i. **DO**

ii. Get the connectivities of the tetrahedron *alltetrahedrons(j)* from the linked list *alltetrahedrons*

iii. If the above triangle parts with the above tetrahedron then $count = count + 1$

iv. $j = j + 1$

v. **WHILE LOOP** until $j \leq j_{last}$

(e) If *count* equals 1 then copy the triangle data in *alltriangles(i)* to *boundarytriangles(k)*

(f) $k = k + 1$, and again set $count = 0$

(g) $i = i + 1$

(h) **WHILE LOOP** until $i \leq i_{last}$

3. ***Algorithm to separate out the triangles belonging to the same boundary, also called as "Stitching"***

Input : Linked list *alltriangles*

Output : Array of linked lists *boundtriangles* of size *totalinternalboundaries* + 1, *totalinternalboundaries*

Steps :

(a) $entity = 1$, $m = 1$

(b) Initiate a linked list *pointsqueue* of type **points**

(c) Copy any one of the node in the first node of the linked list, *boundarytriangles* to *pointsqueue(1)*

(d) Set one to the flag of the node *pointsqueue(1)*

(e) $q = 1$, $i = 1$

(f) **DO**

i. **DO**

- **DO**
  - **IF** the node *pointsqueue*(q) parts with the triangle *boundarytriangles*(i) **then**
    - $k = 1$
    - Copy the nodes of the triangle *boundarytriangles*(i) which is not the same as the node *pointsqueue*(q) and increment q accordingly.
    - Set zero value to the flags of each node newly added to the linked list *pointsqueue*.
    - Make the linked list *pointsqueue* circular and delete the repeating nodes from it.
    - **IF** the flag of *boundarytriangles*(i) is not set **then**

      → Copy the connectivities of the *boundary triangles*(i) to the new linked list of type **triangles** called *boundtriangles*(*entity*)(m).

      → $m = m + 1$

      → Set one to the flag of *boundarytriangles*(i).

* **ENDIF**

- **ENDIF**

• $i = i + 1$

• **WHILE LOOP** until $i \leq i_{last}$

ii. Set zero value to *nodechangeflag*, $i = 1$

• **DO**

- **IF** flag of *pointsqueue*(i) is set to zero **then**

→ The node in *pointsqueue*(i) is next current node to be considered.

→ Set one to the flag in the above node and set the *nodechangeflag*.

- **ELSE** $i = i + 1$

- **ENDIF**

• **WHILE LOOP** until $i \leq i_{last}$

iii. **IF** *nodechangeflag* is set to zero, then discontinue this loop.

iv. **WHILE LOOP** until discontinuing in the previous step.

(g) Set zero value to *trianglechangeflag*

(h) $j = 1$

• **DO**

- **IF** the flag of *boundarytriangles*(j) is set to zero, **then** (this means that there are internal boundaries)

* Restart the linked list *pointsqueue*

* One of the nodes of *boundarytriangles*(j) is taken as the current working node.

* *entity* = *entity* + 1

* Set one to *trianglechangeflag*

* Discontinue from this **WHILE LOOP**

- **ELSE** $j = j + 1$

- **ENDIF**

- **WHILE LOOP** until $j \leq j_{last}$

(i) **IF** *trianglechangeflag* is set to zero **then** discontinue this **WHILE LOOP**

(j) $i = 1,\ q = 1$

(k) **WHILE LOOP** until discontinued from the previous step

(l) *totalinternalboundaries* = *entity* - 1

(m) Array of linked lists *boundtriangles* is made circular for each boundary.

4. ***Algorithm to identify and extract the external boundary and to delete the corresponding linked list from*** *boundtriangles.*

Input : Linked list *boundtriangles*

Output : Linked list of type *triangles* called *extbound*, a node which exists on the convex part of the external boundary called *outmostnode*

Steps :

(a) $i = 1, j = 1$

(b) Initiate a linked list *pointsqueue* of type **points**.

(c) **DO**

- Extract each node from the array of linked list *boundarytriangles*(i) and start storing them in the new linked list *pointsqueue*.
- $i = i + 1$

(d) **WHILE LOOP** until $i \leq i_{last}$

(e) **DO**

- In one another field variable allocated for each node of type **point**, called as *distance*, store the distance of this node from any arbitrary point in space, say the origin, in each node of linked list *pointsqueue*(j).
- $j = j + 1$

(f) **WHILE LOOP** until $j \leq j_{last}$

(g) $j = 1$

(h) Set *outmostnode* as the first node in the linked list *pointsqueue*(j).

(i) **DO**

- **IF** the *distance* of *pointsqueue*(j) is greater than *distance* of *outmostnode* **then** set *outmostnode* as the node in *pointsqueue*(j).
- $j = j + 1$

(j) **WHILE LOOP** until $j \leq j_{last}$

(k) Set zero value to *breakingflag*

(l) **LOOP** from $i = 1$ to *totalinternalboundaries* + 1

i. **DO**

- **IF** *outmostnode* is one of the nodes of *boundtriangles*(i)(j) **then**
  - *externalboundarynumber* = i
  - Set one to *breakingflag*
  - Discontinue from this loop.
- **ELSE** $j = j + 1$
- **ENDIF**

ii. **WHILE LOOP** until discontinuing from the loop.

(m) Discontinue from the **LOOP** if *breakingflag* is set.

(n) Initiate a linked list of type **triangles** called *extbound* and start copying the linked list *boundtriangles*(*externalboundarynumber*) in to it.

(o) Delete the whole linked list *boundtriangles*(*externalboundarynumber*) and rename or copy the remaining linked lists to array of linked list called *intbound*.

5. ***Algorithm to extract all the nodes separately from triangular data contained in the already obtained linked lists***

Input : Linked lists *extbound* and *boundtriangles*

Output : Linked lists *extbdypts*, containing all the nodes on the external boundary, *intbdypts*, containing all the nodes on each internal boundary if any.

Steps :

(a) Initiate an array of linked lists of type **points**, called *intbdypts*, two other linked lists of same type called *extbdypts* and *allpoints*.

(b) **LOOP** from $i = 1$ to *totalinternalboundaries*

- $j = 1$, $k = 1$
  - **DO**
    - Copy all the three nodes of the triangle stored in *boundtriangles*$(i)(j)$ to *intbdypts*$(i)(k)$
    - $k = k + 3$
    - $j = j + 1$
  - **WHILE LOOP** until $j \leq j_{last}$
- Make the linked list *intbdypts*(i) circular and delete the repeating nodes from it.

(c) **LOOP** until termination reached

(d) Repeat the same procedure above except the **LOOP** with the linked list *extbound* as the input to get the new linked list containing all the points on the external boundary called *extbdypts*.

(e) Repeat the same procedure again except the **LOOP** with the linked list *alltriangles* as the input and get the new linked list containing all the points in and on the region called as *allpoints*.

6. ***Algorithm to extract all the nodes inside the region***

Input : Linked lists *allpoints* , *extbdypts* , *intbdypts* .

Output : Linked list of type *points* called *innernodes*.

Steps :

(a) $i = 1, j = 1$

(b) Initiate a linked list of type **points** called as *innernodes*.

(c) **DO**

- Take the node *allpoints*(i).
- Start traversing the linked list *extbdypts* and check whether the node taken above lies in this linked list.
- Set one to *innerflag1* if true, zero otherwise.
- Start traversing the array of linked lists *intbdypts* for each internal boundary and check whether the same node taken above matches with any one of the nodes in these linked lists.
- Set one to *innerflag2* if true, zero otherwise.
- **IF** both the flags *innerflag1* and *innerflag2* are set **then** copy this node in the node of the linked list *innernodes*(j).
- $j = j + 1$

(d) $i = i + 1$

(e) **WHILE LOOP** until $i \leq i_{last}$.

(f) Make the linked list *innernodes* circular.

7. ***Algorithm to extract all the edges all over the region.***

Input : Linked list *alltriangles*.

Output : Linked list of type *edges* called as *alledges*.

Steps :

(a) $i = 1$

(b) Initiate a linked list of type **edges** called as *alledges*.

(c) **DO**

- Take out all the three edges of the triangle stored in *alltriangles*(i) and copy them in to three consecutive nodes of the linked list *alledges*.
- Increment *alledges* accordingly.

(d) $i = i + 1$

(e) **WHILE LOOP** until $i \leq i_{last}$.

(f) Make the linked list *alledges* circular and delete the repeating edges from it.

8. ***Algorithm to orient all the triangles on the external boundary coherently, by making all the normals to point outside the region.***

Input : Linked lists *extbound, alltetrahedrons.*

Output : Modified linked list *extbound* **according to the conventions**

Steps :

(a) $i = 1$

(b) **DO**

- take out the nodes of the triangle *extbound*(i) and store as it is as *N1*, *N2*, *N3*.
- Traverse the linked list *alltetrahedrons* and find out which tetrahedron has the triangle *extbound*(i) with it.
- find out the other innernode of the tetrahedron *N4*.
- Find out the cross product of $\overline{N1N2}$ with $\overline{N1N3}$ and find out the dot product of the resultant vector with the vector $\overline{N4N1}$.
- If the number got after cross product turns out to be negative then interchange *N2* and *N3* from their positions so that they will be swapped.
- $i = i + 1$

(c) **WHILE LOOP** until $i \leq i_{last}$

9. ***Algorithm to make the entire polygon anticlockwise***

Input: *Linked list extbound* **of type edges**

Output : *Linked list alltriangles* **of type triangles**

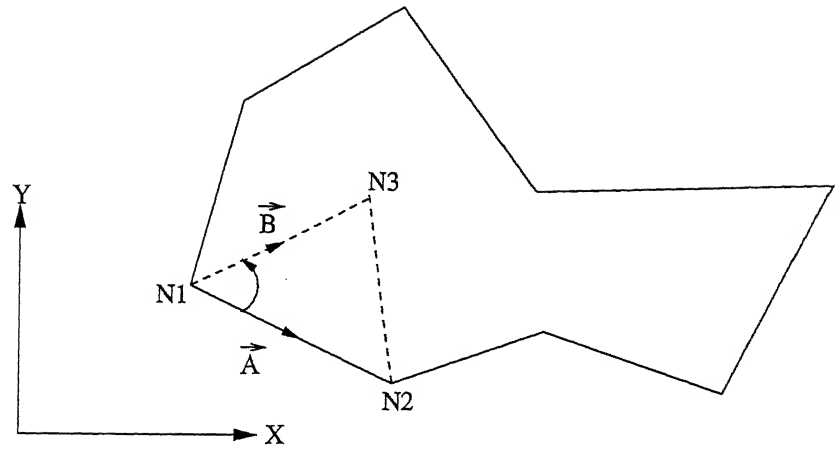


Figure A.1: Making a polygon anticlockwise

Steps :

(a) $i = 1, j = 1$

(b) **DO**

- Let *N1* and *N2* be the nodes stored in *extbound*(i).
- **DO**
  - If *N1* and *N2* are two among the three nodes of the triangle *alltriangles*(j) then let the third node be *N3*.
  - Find out the vector from the node *N1* to *N2*. Let this be $\vec{A}$. Find out another vector from the node *N1* to *N3*. Let this be $\vec{B}$. A schematic representation of this step is shown in the figure A.1
  - Find out the cross product of vectors $\vec{A}$ and $\vec{B}$. If the vector normal to these vectors obtained from the cross product is along negative Z-axis, interchange the positions of *N1* and *N2*.
  - $j = j + 1$
- **WHILE LOOP** until $j \leq j_{last}$.
- $i = i + 1$

(c) **WHILE LOOP** until $i \leq i_{last}$.

10. ***The Algorithm to calculate solid angles for each triangle on the external boundary in the case of 3-d is as follows***

Input : Linked list *extbound*, Centre of the sphere *(Xc, Yc, Zc)* and Radius of the sphere *Rad.*

Output : Solid angles calculated and stored in the field variables of the nodes of the same linked list *extbound* called as *solidangle.*

Steps :

(a) $i = 1$

(b) **DO**

- The vector from the centre of the sphere to any one of the nodes of the triangle *extbound*(i) is calculated. Let this be denoted by $\vec{A}$.
- The normal vector to the triangle in consideration is found. Let this be denoted by $\vec{N}$.
- The dot product of the vectors $\vec{A}$ and $\vec{N}$ is found.
- **IF** the dot product is almost 0 then solid angle of this plane is assumed to be 2π sterdians where in this case the triangle is a major.
- Solid angle is calculated for the triangle in consideration as explained.
- **IF** the solid angle happened to be negative then the solidangle is calculated as that solid angle subtracted from 4π. In this case this triangle is major.
- This solid angle is stored in a new field variable called as *solidangle* in each triangle *extbound*(i).
- $i = i + 1$

(c) **WHILE LOOP** until $i \leq i_{last}$

11. ***Algorithm to calculate the angle of each edge of the polygon in case of 2-d.***

Input : Linked list *extbound* of type edges, Centre of the circle *(Xc, Yc)* and Radius of the circle *Rad.*

Output : Angle of each edge calculated and stored in the field variables of the nodes of the same linked list *extbound* **called as** *edgeangle.*

Steps :

(a) $i = 1$

(b) **DO**

- Take out the nodes of the edge *extbound*(i). Let it be *N1* and *N2.*
- Find out the lesser angle between the lines joining centre of the circle and the nodes *N1* and *N2.*
- Find out the signed area of the triangle formed by the nodes in order *N1,N2* and the centre of the circle.
- **IF** the signed area is negative, the angle of the edge is calculated by subtracting the angle calculated in the previous step from 2π. The angle so calculated is stored in a new field variable called *edgeangle*
- $i = i + 1$

(c) **WHILE LOOP** until $i \leq i_{last}$

12. ***Algorithm to extract all the triangles which share an edge on the boundary.***

Input : Linked list *extbound.*

Output : Linked list *combinations* capable of storing two triangles.

Steps :

(a) $i = 1, j = 1$

(b) Initiate a new linked list called *combinations* which is capable of storing 2 triangles.

(c) Unset all the flags in the nodes of the linked list *extbound.*

(d) **DO**

- Take out the triangle *extbound*(i) and store it in *triangle1.*

- Start traversing the linked list *extbound* again and take out the triangle which edges with *triangle1* and store it in *triangle2*.
- Store these two triangles in *combinations*(j) if flag of the *triangle1* is unset.
- $j = j + 1$

(e) $i = i + 1$

(f) **WHILE LOOP** until $i = i_{last}$

13. ***Algorithm to extract all the edges which are inside the polyhedron***

Input : Linked list *alledges*, **linked list** *extbdypts*.

Output : Linked list *inneredges* **of type edges.**

Steps :

(a) $i = 1$

(b) **DO**

i. Take the edge *alledges*(i).

ii. Start traversing the linked list *extbdypts* and check whether both the nodes of the *alledges*(i) exist in *extbdypts*. Set the flag if true.

iii. **If** the flag is unset then add this edge *alledges*(i) to the linked list *inneredges* .

(c) $i = i + 1$

(d) **WHILE LOOP** until $i \leq i_{last}$

(e) Make the linked list *inneredges* circular.

Bibliography

[1] Barraquand, J., Langlois, B. and Latombe, J.C. *Numerical Potential Field Technique for Robot Path Planning.*IEEE Transactions on System,Man and Cybernetics,22(2)(1992)224-241.

[2] Bohlin, Robert; Kavraki, Lydia E. *Path Planning using Lazy PRM.* Proceedings - IEEE International Conference on Robotics and Automation v1 (2000) 521-528.

[3] Chung C.H; Lee, K.S, *Neural Network Application to the Obstacle Avoidance Path Planning for CIM*, Proceedings of the International Conference on Soil Mechanics and Foundation Engineering v2, (1989),824-828

[4] David E. Goldberg.*Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Pub Co.

[5] Dozier, Gerry; McCullough, Shaun; Holaifair, Abdollah; Tunstel, Eddie; Moore, Loretta, *Multiobjective Evolutionary Path Planning via Fuzzy Tournament Selection.* Proceedings of the IEEE Conference on Evolutinary Computation. (1998) 684-689.

[6] Gerke, Michael, *Genetic Path Planning for Mobile Robots.* Proceedings of the American Control Conference. v4 (1999) 2424-2429.

[7] Han, Woong-Gie; Baek, Seung-Min; Kuc, Tae-Yong, *Genetic Algorithm Based path and Dynamic Obstacle Avoidance of Mobile Robots.* Proceedings of the IEEE International Conference on Systems, Man and Cybernetics. v3 (1997) 2747-2751

[8] Hocaoglu, Cem; Sanderson, Arthur C.*Multi-dimensional Path Planning Using Evolutionary Computation.* Proceedings of the IEEE Conference on Evolutionary Computation. (1998) 165-170.

[9] James Kent, R., Wayne Carson, E and Richard Parent, E. *Shape Transformation for Polyhedral Objects.* Computer Graphics 26(2) (1992) 47-54.

[10] Jason A.Janet, Ren C,Luo, and Michael G.Kay.*Autonomous Mobile Robot Global Motion Planning and Geometric Beacon Collection Using Traversability Vectors.*IEEE Transactions on Robotics and Automation,13(1) (1997) 132-140.

[11] Juidette, H.; Youlal, H.*Fuzzy Dynamic Path Planning using Genetic Algorithms.* Electronics Letters v 36 (n4) (2000) 374-376.

[12] Kalyanmoy Deb, Hans-Georg Beyer.*Self-Adaptive Genetic Algorithms with Simulated Binary Crossover.* Technical report n0.CI-61/99(1999) Department of Computer Science/XI, University of Dortmund, Germany.

[13] Kavraki, L.E., Koulountizakis, M. and Latombe, J.C.*Analysis of Probabilistic Road Maps for Path Planning.*IEEE Transactions on Robotics and Automation,14(1)(1998)166-171

[14] Kavraki, L.E., Motwani, R., Latombe, J.C., and Raghavan, P..*Randomized query processing in Robot Motion Planning.*Journal of Computer and System Science,57(1998) 50-60.

[15] Kavraki, L.E., Svestka, P., Latombe, J.C., and Overmars, M..*Probabilistic Roadmaps for Path-Planning in higher dimensional configuration Spaces.*IEEE Transactions on Robotics and Automation,12(4) (1996) 566-580.

[16] Keerthi S.S; Ong C.J; Huang, E.; Gilbert, E.G. Equi Distant Diagram - *P A New Roadmap Method for Path Planning.* Proceedings - IEEE International Conference on Robotics and Automation v1 (1999) 682-687.

[17] Latombe, J.C. *Robot Motion Planning*, Kluwer Academic Publishers Group,1991.

[18] Mearchou, Andreas C. Path Planning of a Mobile Robot using Genetic Heuristics. Robotica v16 (1998) 575-588.

[19] Milind Bhimrao Joshi, Bhaskar Dasgupta,*Domain Mapping for Path Planning in 3D*. M.Tech thesis submitted to the Department of Mechanical Engineering, Indian Institute of Technology Kanpur, India.

[20] Misashi, Arimoto, Suguru, Parallel-Processable Recursive andHeuristic Method for Path Planning. Proceedings of the International Conference on Soil Mechanics and Foundation Engineering. v2 (1989) 616-618.

[21] Rimon, E., and Koditachek, D.E.*Exact Robot Navigation using Artificial Potential Functions*.IEEE Transactions on Robotics and Automation,8(1992) 501-518.

[22] Suryawanshi, A.B., Bhaskar Dasgupta and Arpan Biswas,*Harmonic Mapping for Path Planning*, submitted to Robotics and Autonomous Systems.

[23] Takeda, H., Facchinetti, C. and Latombe, J.C.*Planning the Motions of the Mobile Robot in a Sensory Uncertainity Field.* IEEE Transactions on Pattern Analysis and Machine Intelligence,16(10)(1994) 1002-1017

[24] Vadakkepat, Prahlad; Tan, Kay Chen; Ming-Liang, Wang, *Evolutionary Artificial Potential Fields and their Application in Real Time Robot Path Planning.* Proceedings of the IEEE Conference on Evolutionary Computation. v1(2000) 256-263.

[25] Vlassis, N.A.; Esanakas, P.*Sensory Uncertainty Field Model for Unknown and Non-stationary Mobile Robots Enviroments.* Proceedings - IEEE International Conference on robotics and Automation. v1 (1998) 363-368.

[26] Wang, Yunfeng; Chirikjian, Gregory S, IEEE International Conference on Robotics and Automation v2 (2000) 977-982.

[27] Warren, Charles W, *A Vector Based Approach to Robot Path Planning.* IEEE International Conference on Robotics and Automation v2 (1991) 1021-1026.

[28] Yuan Y. Tang, Ching Y. Suen,*New Algorithms for Fixed and Elastic Goometric Transformations Models.* IEEE Transactions on Image Processing v3 (n4) (1994) 355-366.

[29] Zhang, Yanjun; Valavanis, Kimon P.*3-D Potential panel method for Robot motion planning.* Robotica v5 (n4) (1997) 421-434.